View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Repositório Aberto da Universidade do Porto

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Sistema de apoio à decisão para a elaboração de mapas de exames no ensino superior

Raphaël Martins Cordeiro

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador: José Fernando Oliveira

Orientador: Maria Antónia Carravilla

27 de Julho de 2017

© Raphaël Cordeiro, 2017

Resumo

A calendarização de exames é um problema típico de otimização combinatória, com a particularidade de ter um número elevado de restrições. A elaboração manual dos calendários é uma tarefa laboriosa, devido justamente ao número de restrições que envolve. A mera estruturação de um calendário que cumpra um pequeno conjunto de requisitos é já bastante demorada, especialmente quando se trata de um número elevado de exames para calendarizar num curto período de exame, em particular porque muitos estudantes estão inscritos em unidades curriculares espalhadas no plano de estudos, como é o caso do ensino superior.

Com o objetivo de automatizar este processo, esta dissertação propõe um modelo e um algoritmo de resolução deste problema, tomando como caso de estudo o curso de Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto.

É apresentada uma revisão bibliográfica sobre o tema, necessária para facilitar a compreensão do problema de calendarização e para enriquecimento pessoal na área da Investigação Operacional.

Após este estudo, foi feito o levantamento dos requisitos, em particular a definição dos objetivos e de todas as restrições do problema, que resultaram numa mistura entre as necessidades específicas do curso e as questões que habitualmente são referidas na literatura. Foi construído um modelo matemático do problema que, depois de validado, serviu de base para o desenvolvimento de uma meta-heurística, BRKGA, *Biased Random-Key Genetic Algorithm*, e implementação. Para tal, foi utilizada uma *framework* já existente, onde se implementaram as rotinas de descodificação e avaliação das soluções, para além da necessária parametrização do algoritmo global. Nestes parâmetros estão incluídos: número de soluções em cada população, número de soluções elite por população, etc. Também para o problema concreto foram definidos parâmetros, como o tempo máximo de intervalo entre exames consecutivos, que é possível alterar.

Após a implementação, foram realizados testes experimentais. Estes testes tiveram como base dados reais relativos ao corrente 2º semestre do ano letivo 2016/2017. Analisados os resultados, foram obtidas soluções extremamente satisfatórias que validaram o objetivo da ferramenta como um sistema de apoio à decisão.

Abstract

Exam timetabling is a tipical combinatorial optimization problem, with the particularity of having a high number of constraints. The traditional manual construction of exam timetables is a laborious task, due to the number of constraints involved. Finding a structure for the calendar that fulfills a small set of constraints is already very time consuming, specially when you're working with a huge set of exams to allocate in a short period of time. This is due in particular to the many students that are attending to classes in different school years, such is the case of higher education programs.

In order to automate this process, this thesis proposes a model and an algorithmh to solve this problem, taking as case study the course of Electrical and Computer Engineering at Faculdade de Engenharia da Universidade do Porto.

A state of the art is presented about the subject, needful to facilitate the understanding of the timetabling problem and for personal enrichment in the field of Operational Research.

After this study, the requirements were surveyed, in particular the definition of the goals and all the constraints of the problem, which resulted in a mix of specific course's needs and issues usually mentioned on the literature. A mathematical model of the problem was built which, after assessment, served as foundation to to the design and development of a meta-heuristic, BRKGA, Biased Random-Key Genetic Algorithm. For such, a framework already existant was used, where the decoder and the solution's evaluation routine were implemented, in addition to de required parameterization of the global algorithm. These parameteres includes: number of solutions in each population, number of elite solutions by population, etc. Also, to this concrete problem, a set of parameteres like maximum time window betweeen consecutive exams are customizable.

After the implementation, several experiments were made. These experiments had as data a real set relative to the current second semester of the academic year of 2016/2017. After the analysis of the results, it is possible to conclude that extremely satisfying solutions were obtained that validated the tool's goal as a decision support system.

Agradecimentos

Em primeiro lugar, queria agradecer aos meus orientadores José Fernando Oliveira e Maria Antónia Carravilla, pelo orientação prestada, pelos seus incentivos, disponibilidade e apoio que sempre demonstraram. Aqui exprimo a minha gratidão.

A todos os meus amigos e colegas, que de uma forma direta ou indireta, contribuíram e auxiliaram na elaboração deste estudo. Pela paciência e atenção que prestaram em momentos mais complicados desta jornada.

Um agradecimento especial à minha namorada, por ter caminhado ao meu lado, pela sua paciência, compreensão e ajuda prestada durante a elaboração deste trabalho. As noites e fins-de-semana que foram sacrificados em prol da dissertação.

Um agradecimento ao meu padrinho, madrinha e avós, que me acompanharam nesta jornada académica e sempre me souberam dar uma palavra de carinho e motivação.

Last but not least, o maior agradecimento vai para os meus pais que me acompanharam nestes 5 anos da minha vida, e sempre souberam prestar o devido apoio, a força e carinho. A eles os dois dedico este trabalho.

A todos o meu sincero e profundo **MUITO OBRIGADO**!

Raphaël Cordeiro

"It never gets easier,
you just go faster"

- Greg LeMond

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

RKGA	Random Key Genetic Algorithm
BRKGA	Biased Random Key Genetic Algorithm
AS	Ant System
ACO	Ant Colony Optimization
SAD	Sistema de apoio à decisão

Capítulo 1

Introdução

Esta dissertação descreve a especificação, conceção, implementação e teste/validação de um sistema de apoio à decisão para a elaboração do mapa de exames no ensino superior. Este capítulo introduz o tema em estudo, qual a motivação e os objetivos do mesmo.

1.1 Motivação

A evolução da tecnologia permite ao ser humano tornar mais eficiente a execução de tarefas complexas, nomeadamente as que envolvem problemas de decisão em que o número de soluções é muito elevado. Esta dissertação insere-se nesta linha ao propor uma solução de automatização de trabalho que, até aqui, é feito de forma manual, de modo a obter resultados similares ou superiores de um modo mais eficiente. O método tradicional de elaboração de mapas de exame baseia-se na análise manual de um conjunto grande de dados e na consequente tentativa de organização dos exames de cada unidade curricular nos períodos de exame previamente definidos, envolvendo um elevado conjunto de recursos humanos e muito tempo.

No ensino superior o número de alunos é elevado, assim como o número de unidades curriculares, não sendo por isso trivial organizar os exames de modo a contemplar as necessidades dos estudantes e a satisfazer as regras previamente definidas. Por exemplo, num caso particular de 80 exames a serem marcados ao longo de 10 dias, cada dia com 3 períodos de exame, o número possível de combinações atinge os $30^{80} \approx 10^{118}$, o que torna clara a necessidade de automatizar este processo.

Apesar deste tema já ter sido abordado por vários investigadores, dada a sua relevância prática, esta dissertação tem como objetivo desenvolver um sistema de apoio à decisão numa perspetiva adaptada às necessidades do curso do Mestrado Integrado em Engenharia Eletrotécnica e de Computadores (MIEEC) da Faculdade de Engenharia da Universidade do Porto, permitindo ao utilizador usufruir da ferramenta para gerar mapas de exame que cumpram um conjunto de regras/restrições e tendo em conta as necessidades dos estudantes.

1.2 Objetivos

Definida a necessidade de desenvolver uma ferramenta eficiente para gerar um mapa de exame, o modelo proposto será implementado com recurso ao algoritmo meta-heurístico BRKGA, propondo-se:

- Respeitar todas e quaisquer restrições que o mapa tem de respeitar;
- Ser capaz de gerar um calendário que seja praticável, na perspetiva do docente;
- Maximizar o intervalo de tempo entre exames;
- Minimizar o número de casos em que há estudantes a ter exames no mesmo dia (a unidades curriculares com pelo menos 2 anos de diferença).

O cumprimento destes objetivos será validado através de um estudo computacional da ferramenta desenvolvida através do uso de uma *framework* para a meta-heurística selecionada.

1.3 Estrutura da Dissertação

Para além deste capítulo, a dissertação possui outros sete capítulos onde:

- **Capítulo 2:** contém uma revisão bibliográfica das diferentes abordagens aos problemas de calendarização em investigação operacional;
- **Capítulo 3:** tem uma descrição detalhada do problema de calendarização de exames;
- **Capítulo 4:** é descrita a meta-heurística BRKGA;
- **Capítulo 5:** é apresentado o modelo proposto para a resolução do problema de calendarização de exames;
- **Capítulo 6:** é descrito o sistema de apoio à decisão desenvolvido para o problema de calendarização de exames;
- **Capítulo 7:** são analisados os resultados dos testes computacionais realizados com o sistema de apoio à decisão desenvolvido.
- **Capítulo 8:** onde serão expressas as conclusões do sistema desenvolvido e indicações para trabalhos futuros.

Capítulo 2

Revisão bibliográfica

2.1 Problemas de calendarização em Investigação Operacional

> *"A combinatorial optimization problem that consists of scheduling a number of examinations in a given set of exam sessions so as to satisfy a given set of constraints."* - Nader Chmait, Khalil Challita

O problema de calendarização faz parte dos problemas mais desafiantes presente em inúmeras áreas, no dia-a-dia. Como **Nader Chmait** e **Khalil Challita** indicam, este é um problema de otimização combinatória, que tem sido abordado por muitas e diferentes metodologias. O problema de calendarização é definido como a alocação de objetos, sujeitos a várias restrições, num período temporal e no espaço de modo a satisfazer um conjunto de objetivos [1].

Como mencionado anteriormente, o problema de calendarização é do tipo NP-difícil, conforme provado por Stephen Cook [2] em 1971, e suportado posteriormente por Richard Karp, em 1972 [3]. Em termos computacionais significa que o tempo de execução de qualquer algoritmo, conhecido até à data de hoje, que garanta a solução ótima, varia de forma exponencial com tamanho do problema em questão.

Nesta secção vão ser apresentados os vários métodos e algoritmos desenvolvidos para este problema em específico e apontadas semelhanças com outros problemas.

2.1.1 Calendarização de exames

Na grande maioria das universidades, institutos e escolas de ensino superior o calendário de exames ainda é feito manualmente, na maior parte das vezes reutilizando soluções elaboradas previamente. O facto de apenas serem alteradas para cumprir algumas restrições, fez com que no caso da FEUP se sentisse cada vez mais a necessidade de recorrer a um método automático, rápido e com pequena percentagem de erro, ou seja, satisfazendo todas ou quase todas as restrições do problema.

A alocação dos exames é então sujeita a várias restrições, que se podem dividir em *hard* e *soft*. As primeiras têm de ser satisfeitas para que a solução seja considerada admissível ou aceitável, enquanto que as *soft* podem ser violadas, mas da forma mais ligeira possível dado que o seu nível de cumprimento representa a qualidade da solução.

2.1.2 Pesquisa Tabu

Pesquisa Tabu, originalmente proposta por Fred Glover, em 1986 [4], consiste num método heurístico que explora a vizinhança da solução, a cada iteração, procurando a melhor solução local e não revisitando uma solução pertencente a uma lista de soluções recentes (lista *tabu*) permitindo evitar um *loop*. Quando uma solução ótima local é encontrada, é comparada com a melhor solução encontrada até ao momento e, se melhor, substitui-a. A lista tabu impede que seja efetuado o procedimento em sentido contrário de modo a, assim continuar-se à procura noutras vizinhanças. O critério de paragem mais habitual para esta abordagem é baseado no conceito de *idle-iteration*, ou seja, para após um número de iterações previamente definido sem melhoria da função objetivo.

Solução Inicial;
while *condição de paragem não cumprida* **do**
    Criar um conjunto de soluções vizinhas;
    Avaliar as soluções vizinhas;
    Escolher a melhor solução admissível;
    Atualizar a lista tabu;
end

Algorithm 1: Algoritmo de Pesquisa Tabu [5]

Aplicação ao problema

Di Gaspero e **Schaerf** [6] levaram a cabo uma investigação sobre técnicas baseadas na Pesquisa Tabu em vizinhanças, incluindo restrições *hard* e *soft*. O tamanho da lista tabu é dinâmica e a função objetivo é adaptável ao longo da procura. Foi demonstrado que a adaptabilidade da função objetivo e uma seleção efetiva das vizinhanças são duas peças-chave nesta abordagem.

Em [7], **Gaspero** aperfeiçoou a abordagem de modo a melhorar a solução obtida com o uso de vizinhanças a partir de uma procura *token-ring* que permite a alteração de exames individualmente (*Recolour*) e com *shakes*, alteração de grupos de exames, com adição de *kickers* que altera a sequência de exames individualmente, conseguindo obter excelentes resultados e reduzindo os tempos de computação.

Restrições consideradas [6]

Simultaneidade (*forte*) – Nenhum estudante pode ter 2 exames marcados para o mesmo *time slot*.

Unicidade (*forte*) – Cada exame apenas pode ser marcado uma e uma única vez em todo o período de marcação dos exames.

Capacidade das salas (*forte*) – Baseado na disponibilidade das salas e no número de carteiras de cada. É necessário que o número de estudantes que estão inscritos para exame seja $\leq$ a capacidade total.

Datas pré-estabelecidas (*forte*) – Exames que devem ser marcados para um certo *time slot*.

Datas proibidas (*forte*) – Exames que não podem ser marcados para um certo *time slot*.

Tabela 2.1: Comparação da função de custo/objetivo entre método de pesquisa tabu genérico e com *Recolour, Shakes e Kickers*

Data set	exams	periods	R, S & K results		TS results	
			best	average	best	average
CAR-S-91	682	35	**5.68**	5.79	6.2	6.5
EAR-F-83	189	24	39.36	43.92	45.7	46.7
HEC-S-92	80	18	10.91	11.41	12.4	12.6
LSE-F-91	381	18	12.55	12.95	15.5	15.9
STA-F-91	138	13	**157.43**	157.72	160.8	166.8
UTA-S-92	638	35	4.12	4.31	4.2	4.5
YOR-F-83	180	21	**39.68**	40.57	41	42.1

Conflitos de primeira ordem (*fraca*) – Penalização segundo uma função de custo que mede a distância entre 2 exames consecutivos de um estudante. À medida que os exames se afastam entre si, o custo decresce.

Conflitos de segunda ordem (*fraca*) – Penalização segundo uma função de custo que conta o número de ocorrências quando um estudante tem um par de exames alocados em *time slots* adjacentes.

1. Penalizar exames em conflito igualmente;
2. Penalizar *overnight*[1] - períodos adjacentes;
3. Penalizar com menor gravidade exames imediatamente antes e depois de almoço.

A **função de custo/objetivo** é do tipo hierárquico, i.e., é uma combinação linear das restrições do tipo *forte* e *fraca* [6] em que o peso considerado para as restrições *forte* é muitissimo maior que o das restantes. A estratégia de atribuição de pesos fixos tem sido considerada ineficaz e para contornar este inconveniente foi decidido que os pesos seriam dinâmicos e iriam variar segundo um mecanismo baseado em [8]:

- Para X iterações consecutivas se todas as restrições de uma componente (*forte* ou *fraca*) forem satisfeitas, então o peso, w, é dividido por um fator γ escolhido aleatoriamente entre 1.5 e 2;
- Para Y iterações consecutivas se todas as restrições de uma componente (*forte* ou *fraca*) forem satisfeitas, então w é multiplicado por um fator γ escolhido aleatoriamente entre 1.5 e 2;
- Para os restantes casos, w é inalterado.

Os parâmetros X e Y fazem parte da parametrização do algoritmo e os seus valores encontram-se entre 2 e 20. Este mecanismo permite à pesquisa Tabu visitar soluções com uma estrutura diferente das visitas anteriormente.

[1] Exame no último período de um dia e exame no primeiro período do dia seguinte.

2.1.3 Arrefecimento simulado

A meta-heurística de pesquisa local arrefecimento simulado, *Simulated Annealing*, tem sido utilizada para problemas de otimização contínuos e discretos [5]. A chave desde método é proporcionar um mecanismo de procura de novos horizontes, para evitar o ótimo local, permitindo *hill-climbing moves*, i.e., movimentos que pioram o valor da função objetivo, procurando deste modo encontrar o ótimo global.

O conceito de arrefecimento simulado foi introduzido em problemas de otimização combinatória por Kirkpatrick et al. [9] e Cerný [10], e é inspirado no processo físico da termodinâmica conhecido por *annealing* [11]. Este processo consiste no aquecimento de um metal e seu posterior arrefecimento controlado, de forma a aumentar o tamanho dos seus cristais e reduzir os seus defeitos.

O arrefecimento é implementado na meta heurística como uma redução da probabilidade de aceitar soluções que pioram a função objetivo. Esta probabilidade é incorporada na função de aceitação e é descrita pela seguinte fórmula:

$$e^{-\frac{\varphi}{T}} \tag{2.1}$$

onde T é um parâmetro de controlo que, na analogia ao processo físico de arrefecimento, corresponde à temperatura, e φ é a diferença entre os valores de função objetivo entre a solução em avaliação e a solução atual. Pode-se observar que à medida que a temperatura diminui, a probabilidade de aceitar novas soluções piores decresce e quando esta atinge o valor 0 apenas se pode aceitar ações melhores, impedindo a escalada de montanha, *hill-climbing* [12].

Segue um exemplo ilustrativo em que a função objetivo é de minimização.

Selecionar solução inicial $i \in S$;
Selecionar temperatura inicial $T_0 > 0$;
Selecionar função de redução de temperatura α;
while *condição de paragem não cumprida* **do**
    Definir contador n=0;
    while *n= número máximo de soluções na vizinhança permitidas a cada temperatura* **do**
        gera j, vizinho de i;
        calcula φ = *f(j) - f(i)*;
        Se $\varphi < 0$ **Então** i:=j;
        Senão *gera número aleatório* $x \in]0,1[$;
        Se x < exp(-φ/t) **Então** i:=j;
        n:=n+1;
    end
    atualiza temperatura;
    T=$\alpha \cdot$ (T)
end

Algorithm 2: Algoritmo genérico de Arrefecimento Simulado [12]

Aplicação para o problema

Em 1998, Thompson e Dowsland [13] desenvolveram uma abordagem constituída por 2 etapas, em que as soluções que cumprem as restrições do tipo *hard* são submetidas a um processo de arrefecimento simulado para melhorar o cumprimento das restrições do tipo *soft*.

Dowsland observou que a forma como a vizinhança estava definida, a importância dos objetivos e a dificuldade dos objetivos eram aspetos fundamentais que afetavam o processo. Dado isto, Burke et al. [14] investigaram a *Kempe chain neighborhood* e concluíram que, para oferecer maior flexibilidade, a manipulação de grupos de exames ao invés de exames individuais trazia melhores resultados.

Foram desenvolvidas outras abordagens baseadas esta meta heurística, e.g. Merlot et al. [15], que recorreu a técnicas de programação por restrições, seguidas de *hill-climbing* de forma a melhorar a solução, usando uma vizinhança *Kempe chain* modificada. Também em [12] se aplicou esta meta heurística ao problema da calendarização de exames.

Restrições *fortes*

- O número de alunos que vão fazer o exame tem de ser menor ou igual à capacidade da sala designada;
- Um estudante apenas pode fazer um exame de cada vez;
- O número de exames designado para um determinado período não pode exceder o número de salas disponíveis;
- Um exame apenas pode ser atribuído a uma única sala.

Restrições *fracas*

- Encurtar o período de exames, com uma penalização crescente à medida que a distância entre exames consecutivos aumenta;
- Um estudante não tem mais do que *um* exame em dois *time slots* consecutivos.

Para além destas restrições, o problema é caraterizado pelos seguintes **atributos**:

1. Existem 4 períodos, *slots*, em cada dia de exame;
2. Número fixo de salas igual a 3;
3. Há 24 exames para planear num total de 2 dias;
4. Salas disponíveis durante o período de exames devem ser utilizadas para atender a um exame.

Em casos específicos, existe a possibilidade de ocorrerem 2 exames na mesma sala, dadas as suas grandes dimensões. Para isso os autores desenvolveram uma função que retorna o número de estudantes que geram um conflito entre esses 2 exames, i.e., o número de estudantes em comum nos 2 exames.

A função de custo é caraterizada pela soma de custos das restrições *hard* e *soft* em que:

- **Custo restrições *fortes* –** é dado através da verificação de estudantes com mais que um exame no mesmo *time slot*, usando a função mencionada anteriormente;

- **Custo restrições *fracas* –** é dado pela multiplicação entre um fator fracionário $\varepsilon \in]0,1[$ e a função de custo das restrições *soft*, também obtido através do retorno da função que analisa os estudantes em comum, para todos os pares de exames que ocorrem em *time slots* consecutivos *no mesmo dia*.

A escolha da temperatura inicial é feita conforme proposto por [16], onde se começa com uma temperatura elevada, que vai ser diminuída rapidamente até que 60% das soluções piores sejam aceites. A temperatura final deste processo será T_0.

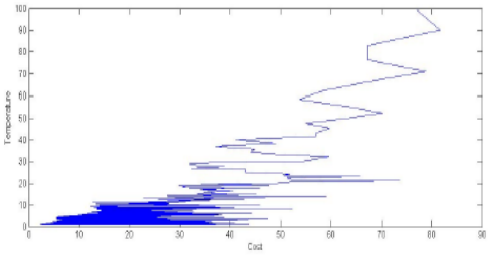
Para decrementar a temperatura ao longo da pesquisa é utilizado o método proposto por Lundy [17], em que para cada iteração é usado um valor de temperatura que é dado pela seguinte equação:

$$T_{i+1} = \frac{T_i}{1+\beta \cdot T_i} \tag{2.2}$$

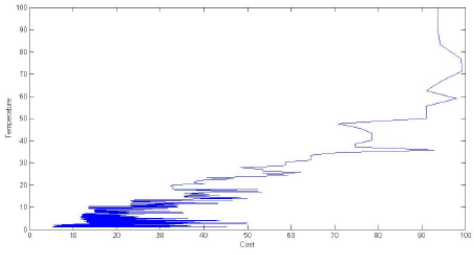
onde β é um valor pequeno (p.e. ≈ 0.0001) e T_i é a temperatura na iteração i. Uma outra solução para o decremento da temperatura é ir alterando dinamicamente o número de iterações à mesma temperatura enquanto o algoritmo evolui, conforme referenciado em [18]

A temperatura final é utilizada como critério de paragem do algoritmo e é definida pelo utilizador. Além desse critério também é feita uma verificação da alteração do custo, em que se este não melhorou em $\frac{T_0}{4}$ iterações o sistema para, pois provavelmente já se atingiu a melhor solução possível para o problema, ou então quando o custo é igual a 0.

À semelhança da heurística Pesquisa Tabu, é também possível criar uma *blacklist* que guarde em memória os dados de vizinhanças onde os valores de custo são demasiado elevados e impraticáveis.



(a) Variação de custos de acordo com o decremento da temperatura



(b) Variação de custos de acordo restringindo vizinhanças com soluções impraticáveis

Figura 2.1: Resultados obtidos com arrefecimento simulado

De acordo com os resultados representados na figura 2.1, embora os valores da função de custo sejam muito semelhantes nos dois casos, é possível distinguir, no caso 2.1b, que a função de custo decresce de uma forma mais rígida que no caso 2.1a, isto devendo-se à capacidade de se esquivar de iterações inúteis.

2.1.4 Colónia de formigas

A ideia da meta-heurística colónia de formigas baseia-se na natureza das formigas reais, na maneira como estas comunicam entre si. Em analogia ao exemplo biológico, ACO [19], *Ant Colony Optimization*, é baseado na comunicação indireta numa colónia de formigas (artificiais).

Estas formigas artificiais constroem soluções candidatas para um problema através da exploração de feromonas artificiais. A comunicação natural numa colónia de formigas (real) é então feita através da modificação do ambiente por distribuição dinâmica de informação, através de feromonas [12].

Os rastos de feromonas (artificiais) são constituídos por informação numérica que as formigas usam para construir soluções, probabilisticamente, e que as formigas adaptam ao longo da execução do algoritmo para refletir a experiência da exploração [5], considerando também a informação heurística acerca do problema a resolver.

As formigas aplicam uma política de decisão local estocástica [2] sempre que se movem. Esta política tem 2 parâmetros: rastos e atração. O processo é o seguinte [20]:

1. Cada formiga, incrementalmente, constrói uma solução para o problema;
2. Quando a solução é completada, a formiga avalia-a e modifica o valor de rasto nas componentes utilizadas;
3. As futuras formigas usam o rasto para próximas explorações.

Uma forma eficaz de verificar o efeito das feromonas é através da experiência de ponte-dupla, ilustrada abaixo:

Como ilustra a figura 2.2, primeiramente, as formigas caminham livremente em direção à comida (ponto F) desde o seu ninho (ponto N). Durante este caminho são depositadas feromonas que refletem o caminho escolhido pela formiga. Outras formigas escolhem aleatoriamente outras alternativas depositando igualmente feromonas. Vai chegar um momento em que um caminho vai estar reforçado pelo rasto de feromonas em grande quantidade, o que significa que foi encontrado o caminho mais curto e, consequentemente, mais rápido.

Outro mecanismo, porém opcional, utilizado em ACO, é *trail evaporation*, evaporação de rastos [12]. Este mecanismo permite decrescer os níveis de rastos a cada iteração do algoritmo de modo a evitar a acumulação ilimitada dos mesmos sobre uma componente, o que leva as chances de a procura ficar estagnada em ótimos locais reduzirem [22].

O primeiro exemplo deste algoritmo é *Ant System* (AS) [23] proposto para a resolução do problema do caixeiro viajante. Apesar de resultados inspiradores, este algoritmo não conseguia

[2] Que contém incerteza, que tem elementos aleatórios.

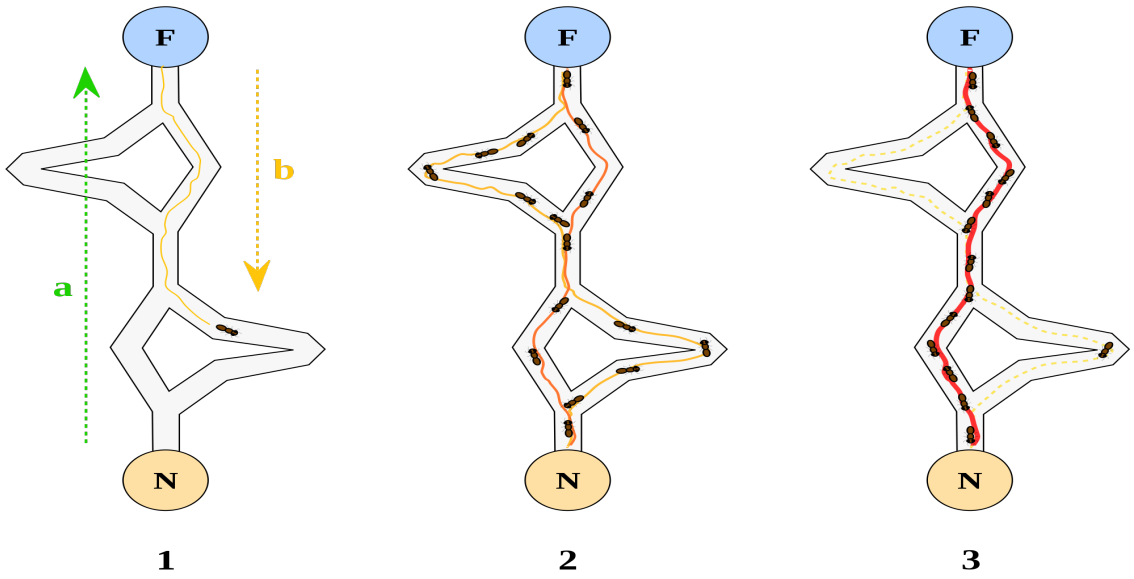


Figura 2.2: Efeito das feromonas [21]

competir com algoritmos "*state-of-the-art*". Contudo, contribuiu para o desenvolvimento de novas perspetivas algorítmicas em investigação futura em diversas outras aplicações, tais como *VRP*, *vehicle routing problem*, *QAP*, *quadratic assignment problem*, calendarização, entre outros.

Em oposição a algoritmos do tipo *greedy* (algoritmo guloso/ganancioso), o ACO permite construir uma vasta variedade de soluções diferentes e, dado isto, explorar um número de soluções elevado [24]. Ao mesmo tempo, o uso de informação heurística, e.g., restrições ao problema, permite guiar as formigas para soluções mais promissoras.

Para problemas de otimização combinatória do tipo estático [3] o algoritmo genérico tem a seguinte estrutura:

Inicializar;
while *condição de paragem não cumprida* **do**
    Construir soluções de formigas;
    Aplicar pesquisa local;
    Atualizar feromonas;
end

Algorithm 3: Esqueleto do algoritmo de otimização colónia de formigas [5]

[3] Problemas estáticos são aqueles aos quais a topologia e os custos associados não se transformam enquanto o algoritmo está em execução. Por exemplo, no problema do caixeiro viajante, as distâncias entre as cidades não se alteram. Por sua vez, nos problemas dinâmicos, a topologia já se altera enquanto as soluções são construídas. Um exemplo de tal problema será o roteamento de redes de telecomunicações, onde os padrões de tráfico se alteram a todo o momento.

Aplicação para o problema

No trabalho [12], já referido anteriormente, é também utilizado um algoritmo baseado em *ACO*, recorrendo também à função *ReturnConflicts(Exam1, Exam2)*, que retorna o número de alunos em comum ao conjunto de exames. Para poder utilizar o algoritmo *ACO*, é criada uma matriz 24*x*24, *PhMatrix* para guardar os valores das feromonas entre exames.

Os valores das feromonas para o problema de calendarização vão ser trabalhados de forma diferente, quando comparado com o problema do caixeiro viajante. Para calcular o custo de viagem entre cidades (de *i* para *j*), no caixeiro viajante, era preciso relacionar apenas a distância, d_{ij}, sem considerar o custo de deslocação. Para o cálculo do custo do calendário de exames, é necessário verificar todos os conflitos entre todos os exames designados para o mesmo *time slot* de cada vez que se adiciona um exame. Desta forma, as formigas irão decidir que exames são passíveis de serem colocados no mesmo *time slot*[4].

Ainda em [12], a matriz *PhMatrix* é inicializada de tal forma que todas os conjuntos $\tau_{ij} : i, j \in$ i,....,n são iguais a 1. A informação heurística é dada por

$$\eta_{ij} = \frac{1}{ReturnConflicts(E_i, E_j)}.$$

A cada iteração, as formigas começam num novo exame e vão construindo o calendário de exames, movendo-se para o próximo exame com maior probabilidade (de acordo com a topologia do problema do caixeiro viajante):

$$p_{ij}^k = \frac{\tau_{ij}^\alpha \cdot \eta_{ij}^\beta}{\sum \tau_{il}^\alpha \cdot \eta_{ij}^\beta} \forall j \in S$$

Na primeira iteração da aplicação do algoritmo, é trivial que o próximo exame escolhido seja aquele que minimiza o número de conflitos, isto é, maximize τ_{ij}, isto porque todas as feromonas têm o mesmo valor.

De modo a evitar a recolocação de um determinado exame no calendário, de cada vez que a formiga *k* escolhe um novo exame, o exame anterior é inserido numa lista similar à *lista tabu*, referida em 2.1.2, que lista todos os exames para onde não pode prosseguir. Verificou-se que escolher, a cada iteração, o exame que contém o menor número de conflitos é eficaz quando se tratam de exames adjacentes, contudo pode levar à geração de conflitos com outros exames colocados na mesma *time slot*. Portanto, escolher o melhor valor possível de η_{ij}, para a marcação de dois exames consecutivos, não conduz necessariamente, à melhor solução global.

Como mencionado acima, as formigas movem-se segundo uma probabilidade e a cada iteração as formigas necessitam fazer uma escolha. Esta escolha tem de ter em conta:

- Considerar todos os exames *j* que não pertencem à *tabu list* da formiga *k*;

[4] É possível ter vários exames ao mesmo tempo desde que o número de salas ocupadas não seja ultrapassado e a capacidade de cada uma delas seja respeitada.

- A soma de todos os valores de feromonas entre os exames já atualmente calendarizados e o próximo exame candidato, $\sum_{l=1}^{i} \tau_{lj}$[5].

A expressão de probabilidade é dada então pela seguinte expressão:

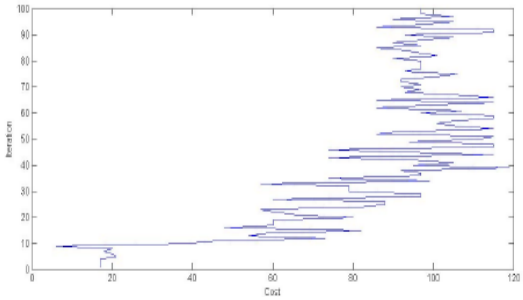
$$p_{ij} = \frac{(\sum_{l=1}^{i} \tau_{lj})^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{z \in S}(\sum_{l=1}^{i} \tau_{lz})^{\alpha} \cdot \eta_{iz}^{\beta}} \forall j \in S \tag{2.3}$$

Assim, a partir do calendário parcial já construído são calculadas todas as probabilidades e custos de movimentação para o exame seguinte. Os passos acima referidos são repetidos até que as formigas completem as suas soluções.

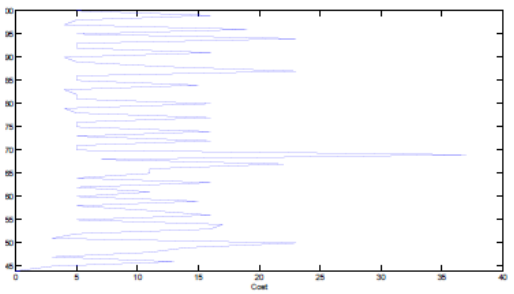
Por fim, falta mencionar como é que as formigas modificam os rastos de feromonas. Depois das formigas terem adicionado um determinado conjunto de exames é feita um *update* das feromonas locais. De modo a evitar distribuições de exames impraticáveis, decidiu-se induzir valores negativos de feromonas entre exames que podem causar configurações de calendário conflituosas. Deste modo, a expressão de atualização do valor das feromonas é dada por:

$$\tau_{ij} = (1 - \phi) \cdot \tau_{ij} - \phi \cdot \tau_{old}, \tag{2.4}$$

onde $\phi \in (0, 1]$ é o coeficiente de decadência de feromonas e τ_{old} é o valor antigo das feromonas.



(a) Variação de custos com o incremento de iterações (calendário muito conflituoso)



(b) Variação de custos com o incremento de iterações (calendário pouco conflituoso)

Figura 2.3: Resultados obtidos com colónia de formigas

A figura 2.3a representa um caso de calendário conflituoso onde a solução ótima é atingida na iteração 10 (quando o custo é menor). A figura 2.3b representa um caso menos conflituoso que o anterior, onde a solução inicial tem um custo de 20 e atinge a solução ótima na iteração 45 com custo igual a zero.

[5] l representa todos os exames já calendarizados até ao momento i.

2.1.5 Algoritmos genéticos

Os algoritmos genéticos, AG, primeiramente introduzidos por Holland [25], são um dos muitos métodos de otimização que usam uma abordagem estocástica na procura de solução, à semelhança de arrefecimento simulado, *hill climbing* e outras abordagens. Ao contrário das metaheurísticas exploradas neste capítulo, que são de pesquisa local, os AG operam sobre uma população de soluções admissíveis para o problema, manipulando-a e evoluindo-a, ao invés de melhorar iterativamente uma única solução.

AG segue uma das ideias de Herbert Spencer *"Survival of the fittest"*, a sobrevivência do mais apto [26], inspirado pela teoria da evolução de Darwin. Baseando-se no fenómeno de evolução biológica, defende-se que a mutação é o princípio da evolução/adaptação.

As soluções são codificadas como cromossomas, sequência de genes/informação, que são evoluídas segundo as operações de *cruzamento* e *mutação*, com o intuito de serem sucessivamente melhores em cada geração de população. O algoritmo genérico consta nos seguintes passos:

Inicializar população;
while *condição de paragem não cumprida* **do**
    Avaliar *fitness* da população;
    Processo de seleção;
    Cruzamento de cromossomas;
    Mutação;
end

Algorithm 4: Algoritmo genérico de algoritmos Genéticos [5]

Cruzamento - operação que permite a 2 cromossomas, designados "pais", permutar os seus genes dando origem a 2 cromossomas descendentes, "filhos".

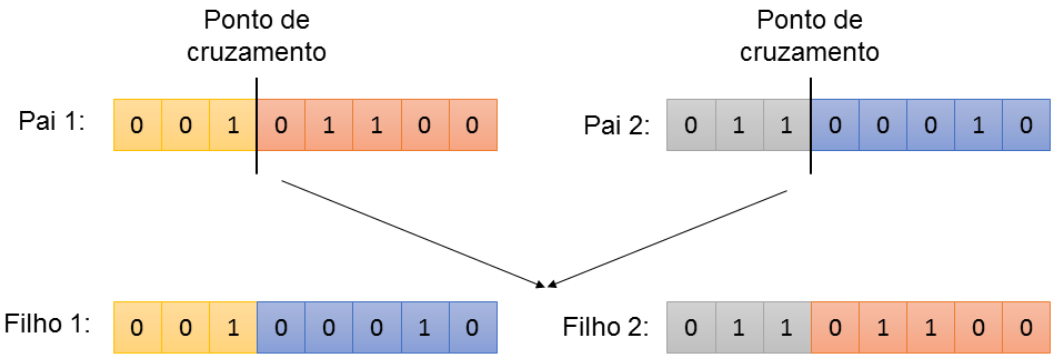
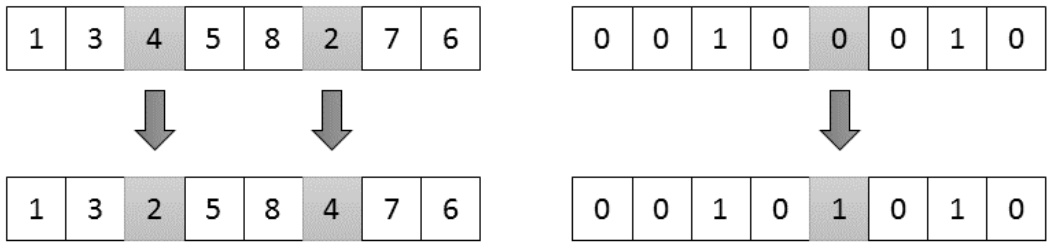


Figura 2.4: Processo de cruzamento em algoritmo genético

Mutação - operação que altera, aleatoriamente, uma posição do cromossoma. Nos casos de codificação binária é feito o complemento do *bit* escolhido.



(a) (b)

Figura 2.5: Processo de mutação em algoritmo genético, (a) Caso variáveis discretas (b) Caso variáveis binárias

Aplicação ao problema

Em [27], Čupić et al. definem os seguintes parâmetros:

- $T = \{t_1, t_2, t_3, .., t_n\}$;
- $S = \{s_1, s_2, s_3, .., s_n\}$;
- $C = \{c_1, c_2, c_3, .., c_n\}$;
- $FXC = \{(c_x, t_y), (c_w, t_z), ..\}$;
- $LC = \{LC_1, LC_2, ..\}$;
- $LC_i = \{c_{i1}, c_{i2}, ..\}$.

T representa os períodos em que podem ser marcados os exames $\in C$, que vão ser feitos pelos estudantes $\in S$. Define-se o conjunto FXC, que representa exames que têm de ocorrer num certo período, independentemente dos restantes exames. O conjunto $LC \in C$, representa os subconjuntos de exames que têm de ocorrer em simultâneo, para isso, estes subconjuntos não podem ter elementos em comum, ou seja, para $i \neq j$, $LC_i \cap LC_j = \emptyset$. De igual modo, é necessário garantir que estes subconjuntos nunca ocorram em simultâneo, ou seja, no caso de subconjuntos associados a um período: (LC_i, t_i) e (LC_y, t_y), só é possível se $i \neq y$.

Cada período pode ser escolhido pelo algoritmo, no entanto, se for alocado um exame que pertence a um subconjunto de LC, todos os exames que pertencem a esse subconjunto devem igualmente ser alocados nesse mesmo período.

Para poder alocar um grupo de exames no mesmo período, $t_i \in T$, é necessário garantir que os estudantes inscritos aos exames que ocorrem em simultâneo não tenham exames sobrepostos, ou seja, para $c_i \in LC_i, \forall\, LC_i \subset LC$, $\bigcap_{i=1}^{n} f(c_i) = \emptyset$, sendo $f(c_i) \subset S$ que representa o conjunto de estudantes que vão atender ao exame c_i.

Restrições *Fortes*:

Foram acima referidas as restrições *fortes* da abordagem ao problema, resumindo:

- Exames pertencentes ao conjunto FXC têm de ser obrigatoriamente alocados a um período em específico;
- Os exames pertencentes ao subconjunto LC_i têm de ser todos alocados no mesmo período e não podem ter estudantes em comum;

- Os subconjuntos de *LC* têm de ser alocados em períodos distintos;
- Os estudantes que vão atender aos exames pertencentes aos subconjuntos LC_i apenas podem estar inscritos a um único exame pertencente a esse subconjunto.

Restrições *Fracas*:

O cumprimento das restrições *fracas* pode não ser suficente para garantir soluções com um nível de qualidade suficiente. Para obter soluções de maior qualidade é necessário avaliar as soluções segundo alguns indicadores adicionais, designadamente:

- O número de estudantes que têm atribuídos mais que 1 exame por dia, assim como o número de ocorrências desta situação;
- O número de estudantes que têm atribuídos exames em períodos adjacentes, assim como o número de ocorrências desta situação.

De notar que nesta abordagem, ao contrário das mencionadas nas secçoes anteriores, é considerado que o caso em que um estudante tem mais de 1 exame por dia faz parte das restrições *soft*.

Cruzamento e Mutação

Nesta abordagem, o cruzamento é feito com uma metodologia diferente da explicada em 2.1.5 e tem os seguintes passos:

1. Os genes do primeiro pai são copiados para o cromossoma do filho;
2. Para cada exame associado a um termo é atualizado para corresponder ao termo no segundo pai com uma probabilidade de 50%;

No caso da mutação, são implementadas 2 operações:

1. Denominado de *course mutation*, que com uma probabilidade aleatória seleciona um novo período.
2. Denominado de *term mutation*, que com uma probabilidade aleatória seleciona um segundo novo período e os exames associados são permutados.

Estrutura do cromossoma

Uma solução pode ser representada de muitas maneiras diferentes. A estrutura genérica, a mais simples, é dada por uma sequência de bits que torna as operações de cruzamento e mutação mais simples. Os autores optaram por uma estrutura diferente, que já provou ter excelentes resultados em [28], que permitiu implementar as operações do algoritmo com ordem de complexidade O(1).

KCourse é uma estrutura simples (mas específica) do cromossoma que contém um apontador para a informação global do exame, índice do período em que é alocado e e apontadores para o anterior e seguinte KCourse, o que permite a criação de uma ligação dupla com grupos de exames que pertencem ao mesmo período.

KCourse[] *kcourse;*
KTerm[] *kterms;*
int[] *clusterTerms;*
int[] *eval;*

Figura 2.6: Cromossoma

KTerm, igualmente uma estrutura do cromossoma, contém a informação global de um período e um apontador para alguns objetos de KCourse calendarizados para esse período. Com estas estruturas definidas, as respostas às perguntas "Exames atribuídos ao período t_i[6]"ou "Em que período está alocado o exame c_i^6"são obtidas diretamente sem necessitar de pesquisa, construindo com maior facilidade o mapa de exames.

Segue-se um exemplo ilustrativo da estrutura do cromossoma na figura 2.7.

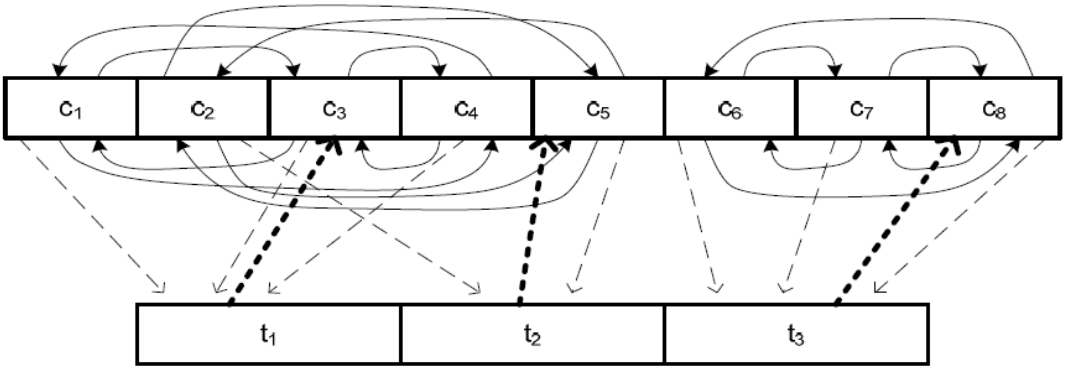


Figura 2.7: Estrutura do cromossoma [27]

Na figura 2.7, as setas de cima representam o KCourse seguinte, enquanto que as setas de baixo representam o KCourse anterior. As setas tracejadas a negrito representam os apontadores dos períodos para os exames e as setas a tracejado simples os apontadores dos exames para os períodos correspondentes. O vetor *clusterTerms* contém os índices que para cada grupo de exames aponta para o período ao qual o grupo está associado. O vetor *eval* contém as componentes de avaliação do cromossoma apresentado de seguida.

Fitness

Como todas as meta-heurísticas, as soluções são avaliadas segundo uma função objetivo, neste caso denominado de *fitness*. Nesta abordagem, a avaliação é composta por 3 componentes. Na primeira componente é avaliado cada exame alocado em cada período, fazendo-se uma contagem do número de estudantes que têm vários exames associados ano mesmo período. A contagem é armazenada numa posição de um vetor contido no cromossoma da solução, eval[0].

[6] Varíavel do problema.

A segunda componente consiste na análise em cada período t_i, e cada exame c_i pertencente a este período, contado-se o número de exames que os estudantes inscritos ao exame c_i têm no mesmo dia, com um fator multiplicativo igual a 4. A esta contagem é-lhe adicionada o número de exames que os estudantes têm no dia seguinte, sem fator multiplicativo. Este valor final é igualmente armazenada no cromossoma, eval[1].

Por último, para cada período é tomado em conta o número de estudantes que excede os limites de capacidade. A soma final é armazenada em eval[2].

Uma solução é definida como admissível, ou praticável, apenas se eval[0] e eval[2] tiverem valor 0.

Procedimento de seleção de pai

De modo a assegurar um ritmo de evolução satisfatório, aquando da seleção de pais para cruzamento são selecionados aleatoriamente 3 soluções e ordenadas pela sua *fitness*. Como as componentes de avaliação de uma solução estão presentes no cromossoma sob a forma de um vetor, são consideradas duas metodologias de seleção:

1. ***wsel*** - de modo a obter um valor de escalar de avaliação dado por atribuição de pesos, p.e., $fitness = (eval[0] + eval[1]) * w_1 + eval[2] * w_2$, onde as soluções são posteriormente ordenadas pelo seu valor escalar;

2. ***hsel*** - corresponde a uma hierarquização por componentes de avaliação em que as três componentes são transformadas em apenas duas $(e_1, e_2) = (eval[0] + eval[2], eval[1])$, onde as soluções são ordenadas pelo primeiro elemento e em caso de igualdade, pelo segundo elemento.

Procedimento de seleção de substituto

Depois de serem criados os cromossomas-filho, é necessário substituir uma solução da geração anterior por estas novas soluções. Os autores proposeram dois procedimentos:

1. ***nrep*** - seleciona a 3ª solução, considerada a menos valiosa, candidata a ser selecionada como pai;

2. ***hrep*** - divide a população em 4 quadrantes baseando-se nos valores do par (e_1, e_2).

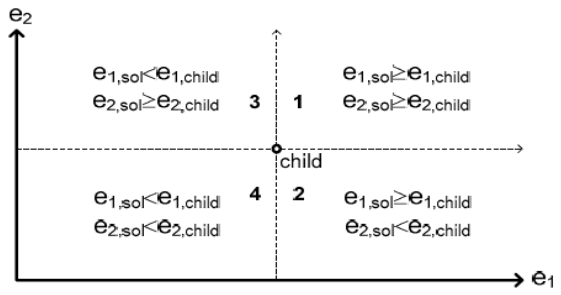


Figura 2.8: Divisão da população

No caso de *hrep*, depois de dividida a população, é achado o primeiro quadrante não vazio (a começar pelo quadrante 1 como ilustra a figura 2.8) e seleciona uma solução aleatoriamente para efetuar a substituição [29].

Tabela 2.2: Resultados obtidos através de algoritmo genético com variação de parâmetros

Population size	pMut [%]	pSwap [%]	Selection	Replacement type	Number of conflicts eval[0]	Solution quality eval[1]	Capacity violations eval[2]
Best 5 solutions (mod3)							
200	1	5	wsel	nrep	0.67	1754.67	0
50	1	1	wsel	nrep	0.33	1763.33	0
20	1	5	hsel	hrep	0.00	1770.00	0
20	1	1	hsel	hrep	0.00	1789.00	0
50	1	5	wsel	nrep	0.33	1795.00	0
Best 5 solutions (mod1)							
100	1	5	hsel	nrep	0.00	1733.33	0
20	2	5	hsel	nrep	0.00	1733.33	0
20	2	2	hsel	nrep	0.00	1749.00	0
200	2	2	hsel	hrep	0.00	1757.67	0
50	2	5	hsel	nrep	0.00	1760.67	0
Best 5 solutions (mod2)							
50	1	5	hsel	hrep	0.00	1708.00	0
20	1	5	wsel	nrep	0.33	1718.67	0
100	1	5	hsel	hrep	0.00	1740.33	0
50	1	5	hsel	nrep	0.00	1742.67	0
20	1	5	hsel	nrep	0.00	1762.33	0

A tabela 2.2 contém os resultados das experiências seguindo a abordagem descrita ao longo desta secção. A 2ª coluna, **pMut**, representa a probabilidade de seleção de um período na operação de mutação *course mutation* e a 3ª coluna, **pSwap**, a probabilidade de seleção de um período na operação de mutação *term mutation*. Na 4ª e 5ª coluna são representados as metodologias de seleção dos cromossomas pai e os cromossomas a serem substituidos, respetivamente.

Os testes foram corridos sobre dados reais, alocação de 77 exames em 30 períodos numa janela temporal de 10 dias. Foram efetuadas 4 000 000 iterações em cada teste. Como fatores de peso no método de seleção *wsel* foram considerados w_1=1 e w_2=0.05.

Durante os testes, foram observadas situações em que alguns períodos t eram pouco selecionados (em *mod1*). De de modo a contornar essa adversidade verificou-se qual a frequência de uso para cada período e modificou-se a função de probabilidade de modo a que o gerador de números retornasse índices de períodos utilizados menos frequentemente com maior probabilidade (*mod2*). Foi ainda implementado uma 3ª fase de testes em que é utilizado *mod1* em 50% do tempo e *mod2* no tempo restante.

Quanto à análise de resultados, é trivial a observação que o método de seleção *hsel* é o único onde se cumpre os requisitos impostos na avaliação das soluções e onde se encontra a melhor solução entre os testes realizados com *fitness*=1708.00.

2.2 Resumo comparativo

2.2.1 Pesos das restrições

Da exposição anterior pode-se concluir que geralmente as restrições fracas permitem dar qualidade ao calendário e, por norma, têm um peso associado inferior às restrições fortes. Como já referido, estas últimas devem ser cumpridas integralmente.

Há autores que optaram por fazer uma gestão dos pesos das restrições, outros que decidiram ter pesos fixos ou não considerar de todo os pesos. O interesse dos pesos dinâmicos é permitir adaptar a função de custo de modo à pesquisa visitar regiões do espaço de soluções que não seriam visitadas de outra forma, criando a possibilidade de obter melhores resultados [6].

Tabela 2.3: Métodos de gestão de pesos abordados pelos autores.

Métodos	**Pesquisa Tabu**	**Arref. Simulado ≡ Colónia de Formigas**	**Algoritmo Genético**
Pesos tomam valores diferentes ao longo do algoritmo.	✗		
Pesos fixos, fracionários, inalterados ao longo do algoritmo.		✗	
Sem considerar pesos, apenas função de custo.			✗

2.2.2 Restrições fortes

A tabela 2.4 permite obter uma visão sobre as diferentes abordagens realizadas pelos autores quanto à escolha das restrições fortes.

Tabela 2.4: Restrições fortes utilizadas pelas diferentes abordagens

Restrições fortes	**Pesquisa Tabu**	**Arrefecimento Simulado ≡ Colónia de Formigas**	**Algoritmo Genético**
Nenhum estudante deve ter mais que 1 exame no mesmo período.	✗	✗	
Cada exame é marcado uma única vez.	✗		
Respeitar capacidade das salas.	✗	✗	
Exames com data fixa devem ser alocados no período correspondente.	✗		✗
Períodos isentos de exames.	✗		
Número de exames marcados para um período, não pode exceder o número de salas.		✗	
Cada exame é-lhe atribuído uma única sala.		✗	
Conjunto de exames a ser marcados num só período.			✗
Conjuntos de exames diferentes alocados em períodos diferentes.			✗
Aluno só atende a 1 exame de um conjunto de exames a ocorrer no mesmo período.			✗

2.2.3 Restrições fracas

A tabela 2.5 permite obter uma visão sobre as diferentes abordagens realizadas pelos autores quanto à escolha das restrições fracas.

Tabela 2.5: Restrições fracas utilizadas pelas diferentes abordagens

Restrições fracas	**Pesquisa Tabu**	**Arrefecimento Simulado ≡ Colónia de Formigas**	**Algoritmo Genético**
Penalização decrescente da distância entre exames consecutivos à medida que estes se afastam.	✗		
Nenhum estudante deve ter mais que 1 exame no mesmo dia.			✗
Nenhum estudantes deve ter 2 exames em períodos adjacentes	✗	✗	✗
Penalização crescente da distância entre exames consecutivos à medida que estes se afastam.		✗	

Capítulo 3

Descrição do problema, conforme se apresenta no estudo de caso.

O problema de calendarização de exames tem sido estudado por muitos investigadores, dado aliar um inegável interesse prático a um apreciável nível de complexidade. Como já referido no capítulo 1, na prática a geração dos mapas de exames é feita de forma manual, segunda uma filosofia de "tentativa-erro", o que torna o processo bastante desgastante e muito longo. No entanto, e como também é referido na revisão de literatura no capítulo 2, resultados interessantes têm sido conseguidos através da utilização de meta-heurísticas. Sendo um problema com um número muito elevado e diversificado de restrições, em situações reais a simples obtenção de uma solução admissível é extremamente difícil, moroso ou mesmo impossível.

O caso real que motivou a elaboração desta dissertação, a calendarização dos exames do MIEEC da FEUP, insere-se na situação acima descrita, sendo por isso um adequado caso a estudar e com o qual validar a abordagem proposta. No resto deste capítulo serão descritas as características e parâmetros concretos desde caso.

As restrições dividem-se portanto em restrições do tipo forte e do tipo fraco. As primeiras, as fortes, são restrições invioláveis que têm de ser necessárias cumpridas, pois a violação das mesmas torna o planeamento não admissível. As restantes são passíveis de serem violadas, são flexíveis, pois não constituem um impedimento à marcação dos exames. Contudo é desejável minimizar este tipo de transgressões.

Esta classificação divide a resolução do problema de calendarização de exames em 2 partes. A primeira, consiste na aplicação do algoritmo de modo a assegurar uma possível solução, respeitando as restrições fortes, de primeira ordem. A segunda parte consiste na otimização do problema, ou seja, encontrar a melhor (ou aproximadamente melhor) solução possível, minimizando o número total de conflitos.

3.1 Parâmetros

Nesta secção serão introduzidos os parâmetros, eventuais requisitos e informações, recolhidos a priori e que é necessário ter em conta para a modelização do problema.

De referir que existem informações que não têm implicação direta no problema, sendo contudo importantes na sua conceção.

Unidades Curriculares

As unidades curriculares podem-se dividir em 3 grupos diferentes, quanto ao seu método de avaliação:

- Unidades curriculares apenas com avaliação distribuída (F);
- Unidades curriculares pendentes de exame final, com acesso a exame de recurso (E+R);
- Unidades curriculares avaliadas com avaliação distribuída com acesso a exame de recurso (F+R).

As unidades curriculares que apenas contêm avaliação distribuída não representam nenhuma participação no calendário de exames e portanto são completamente excluídas do processo. No entanto, as restantes já têm de ser consideradas. Para o 3º caso apresentado acima, a componente F será tratada como um exame (E).

Turnos - *Time Slots*

A marcação dos exames restringe-se a estes três horários, disponíveis ao longo de cada dia útil:

- Turno das 09:00;
- Turno das 13:30;
- Turno das 17:00.

Existe a possibilidade de se poder marcar vários exames para o mesmo turno desde que estes não tenham estudantes em comum. De notar que os exames não podem ser marcados para datas referentes a **fins de semana (Sábado e Domingo) ou feriados**.

Recursos físicos

Os recursos neste caso referem-se às salas disponíveis para a realização de exame:

- Salas de informática;
- Salas de laboratório;
- Salas de desenho;

- Salas de prova escrita;

No caso em estudo os dados referem-se ao curso MIEEC, sendo que só serão consideradas as salas de informática e de prova escrita. É considerada a capacidade real das salas durante a execução do algoritmo.

Sobreposições

É imperativo que não exista sobreposição de exames, i.e., dois exames marcados para o mesmo turno, no mesmo dia com alunos em comum. Como referido anteriormente, apenas exames mutuamente exclusivos podem ser marcados para o mesmo turno, no mesmo dia.

Prioridade de uma unidade curricular

No caso de conflito entre exames, pode ser dada prioridade ao exame pertencente à unidade curricular:

- Com um grau de complexidade superior (através do número de créditos/taxa de aprovação);
- Com um número de alunos substancialmente superior.

Intervalo entre exames/épocas de exame

Ter em conta que para exames de unidades curriculares do mesmo ano será necessário um intervalo mínimo de 4 dias entre exames. Da mesma forma, é considerado um intervalo de pelo menos 3 dias entre épocas de exame. Para verificar estes pormenores são construídos conjuntos de exames, *clusters*, que ocorrem sempre em conjunto no mesmo ano curricular.

Os parâmetros acima representados são os pontos cruciais na elaboração do problema a considerar ao longo do trabalho. Em suma, o problema caracteriza-se por pretender estabelecer exames em diferentes turnos para cada dia do calendário de avaliações, de maneira a que sobreposições não sejam criadas assim como dando sempre resposta aos requisitos e restrições previamente enunciados.

3.2 Objetivos

O objetivo considerado neste trabalho é nada mais que maximizar o tempo entre exames, de modo a serem cumpridas um conjunto de restrições.

3.3 Restrições

Como referido anteriormente, este problema está sujeito a um vasto número de restrições que são classificadas de fortes ou fracas. As restrições consideradas para a resolução deste problema são as seguintes:

Restrições Fortes- Restrições que devem ser cumpridas para ser possível aceitar uma solução:

- Não podem ocorrer mais que 3 exames no mesmo período;
- Exames pertencentes ao mesmo período não podem ter estudantes em comum;
- Exames em períodos adjacentes não podem ter estudantes em comum;
- Exames do mesmo ano não podem ocorrer no mesmo dia;
- Exames de anos consecutivos não podem ocorrer no mesmo dia;
- Garantir que, para qualquer calendário possível, o primeiro exame de época de recurso inicia-se posteriormente ao último exame de época normal.

Restrições Fracas- Restrições de nível secundário que são passíveis de ser transgredidas:

- Cada estudante tem no máximo 1 exame por dia;
- Intervalo de tempo entre a época normal e a época de recurso, dentro dos limites;
- Intervalo de tempo entre exames (época normal e recurso) pertencentes ao mesmo *cluster*, dentro dos limites.

Capítulo 4

Introdução ao BRKGA

4.1 Introdução

Neste capítulo é introduzida a meta-heurística BRKGA, *Biased Random-Key Genetic Algorithm*, a metodologia que vai ser implementada neste problema. No que respeita à organização do capítulo, na primeira secção será introduzida a estrutura genérica da heurística e numa segunda secção, uma visão mais detalhada do funcionamento da mesma em problemas de otimização combinatória.

4.2 A meta-heurística BRKGA

BRKGA é uma meta-heurística evolutiva baseada no algoritmo genético de chaves aleatórias de Bean [30], RKGA, *Random-Key Genetic Algorithm*, desenvolvida para abordar problemas de sequenciamento e otimização em geral, o que torna o uso desta ferramenta muito interessante para a resolução do problema, visto este se tratar de sequenciamento de exames. No RKGA, os cromossomas são representados por um vetor de número reais aleatoriamente gerados $\in [0, 1)$. O algoritmo determinístico, denominado de descodificador, recebe como entrada um cromossoma que contém uma solução para o problema de otimização que vai ser avaliado através de uma função objetivo para determinar o seu *fitness* [31].

Bean prôpos o RKGA para resolver 3 tipos de problemas de otimização: designação quadrática[1], programação de tarefas em máquinas em paralelo[2] e alocação de recursos. No entanto, refere que o RKGA possui tal flexibilidade que, com poucas adaptações, pode abranger diversos problemas como caixeiro viajante, roteamento de veículos, *knapsack problem*[3] entre outros.

[1] Designação quadrática - consiste em posicionar um conjunto de instalações em um determinado conjunto de localidades, dados os fluxos entre as instalações e as distâncias entre as localidades, com o objetivo de minimizar o somatório que liga os fluxos às distâncias de acordo com a distribuição feita.

[2] Para minimização de atraso total.

[3] Knapsack Problem - problema da mochila.

O algoritmo BRKGA, denominado por J. Gonçalves e M. Resende [31], em distinção ao RKGA, utiliza um mecanismo tendencioso na seleção dos cromossomas para a operação de cruzamento. Este mecanismo seleciona um dos cromossomas candidatos, aleatoriamente, do grupo elite (melhor fitness) e outro pertencente ao grupo não-elite aleatório para serem cruzados e originar uma nova solução. No modelo de Bean, a escolha de candidatos é feita aleatoriamente sem ter em consideração grupos elitistas, apesar de os considerar em outras operações.

Desta forma, no BRKGA, a probabilidade de uma nova solução herdar genes de um pai *elite* é muito maior, visto que a probabilidade de um cromossoma elite ser escolhido é $\frac{1}{p_e}$ e $p_e < \frac{p}{2}$, enquanto que no RKGA a probabilidade de escolher uma solução elite é de $\frac{1}{p}$. Esta pequena diferença entre as duas metodologias permite que o BRKGA seja superior ao RKGA [32].

4.2.1 Estrutura e Operações

Os mecanismos que interligam o BRKGA ao problema são:

- **Codificação**- a solução é apresentada sob forma de um cromossoma;
- **Função Objetivo**- a medição da qualidade da solução, em cada cromossoma.

A codificação da solução é o que permite transcrever a solução do problema para o utilizador de modo a poder construir o mapa de exames.

Como exposto na secção 2.1.5, a reprodução da população em algoritmos genéticos é feita através das operações de cruzamento e mutação de soluções. Contudo, esta abordagem aplica essas operações de uma maneira diferente com uma sequência de passos diferente do tradicional algoritmo genético.

A divisão elitista permite distinguir um conjunto de soluções de outras pelo seu valor, de acordo com uma função objetivo definida, em que uma fração das melhores soluções pertencem ao grupo elite e as restantes ao grupo não elite. Em termos práticos para uma população de 20 soluções, p, e se definir p_e, a população elite, com capacidade para 5 soluções, as melhores 5 soluções pertencem a essa população e as restantes 15 à população não elite, sendo que $p_e < p - p_e$ ou $p_e < \frac{p}{2}$.

Um conceito adicional à reprodução de uma nova população advém deste processo de partição. Todos as soluções $\in p_e$, da geração k são copiadas diretamente para a nova geração, $k+1$[4]. A geração $k+1$ vai incluir, além das soluções elite, o resultado do cruzamento de soluções e mutações, detalhadas em seguida.

Assim, uma nova geração, com população p, contém as soluções elite da geração anterior, p_e, soluções obtidas através da mutação, p_m, e soluções geradas através de cruzamento, $p - p_e - p_m$, tal como ilustra a figura 4.1.

Em contrapartida ao processo de mutação genérico exposto em 2.1.5, a mutação não é aplicada sobre as soluções geradas através de cruzamento. No BRKGA são geradas soluções de forma aleatória, do mesmo modo como a população inicial é gerada, fazendo parte do conjunto p_m,

[4]É esta etapa do processo de criação de uma nova geração que a heurística RKGA usa o conceito de grupo elitista

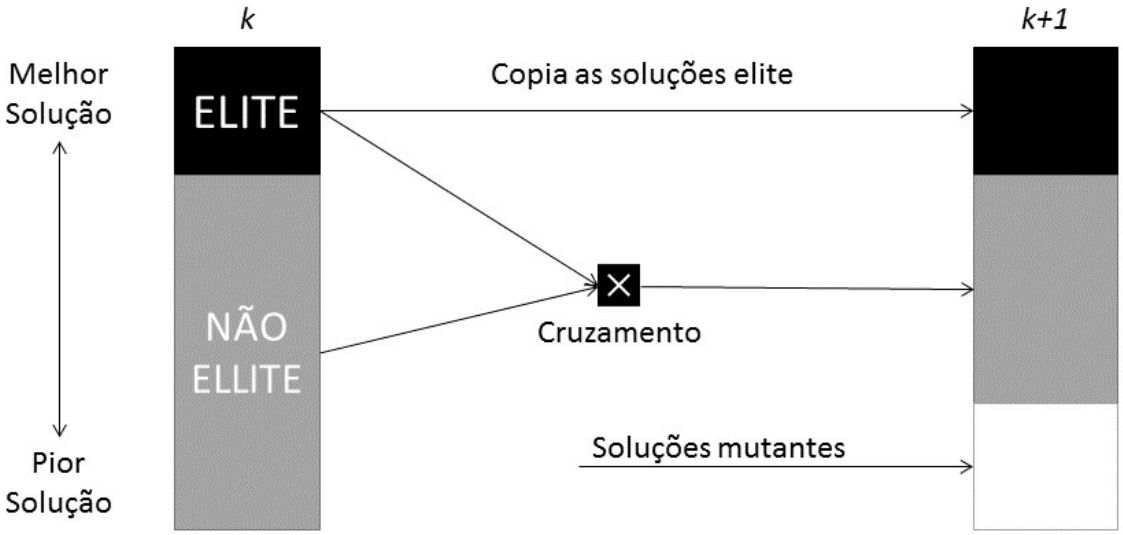


Figura 4.1: Transição da geração *k* para a geração *k+1*.

população de mutantes que têm como missão aplicar o conceito de evolução, baseado na teoria de Darwin.

Quanto à operação de cruzamento, a metodologia é também distinta do cruzamento genérico apresentado na figura 2.4, no capítulo 2, secção 2.1.5 onde é escolhido um ponto de referência para permutação dos genes. Esta abordagem é denominada de cruzamento uniforme parametrizado, introduzido por Spears e DeJong em 1991 [33], é aqui que o conceito *random-keys*, chaves aleatórias, entra.

Este processo é similar ao lançamento de um dado: seis faces, com probabilidade de $\frac{1}{6}$, caso não esteja viciado, cada uma de ficar voltada para cima. No entanto, existem dados viciados, *biased*, ao qual a probabilidade de sair um certo lado é superior aos restantes. Igual acontece com o atirar de uma moeda, tendo esta 2 faces com probabilidade de $\approx 50\%$ cada uma de ficar virada para cima, existem também moedas viciadas em que pode haver uma probabilidade de 90% em sair determinada face.

No processo de cruzamento, em analogia aos casos descritos acima, é definido pelo utilizador uma probabilidade de para um dado gene, *i*, do cromossoma descendente este herdar o gene correspondente na solução elite escolhida para reprodução.

Considera-se ρ_e, como a probabilidade de uma solução descendente herdar o gene da solução elite. Por sua vez, uma solução descendente tem probabilidade de 1-ρ_e de herdar genes da solução não elite.

Dito isto, para poder determinar quando é que a solução herda ou não genes da solução elite é feita uma comparação entre cada posição de um vetor de chaves aleatórias reais geradas dentro do intervalo [0,1[e o valor de ρ_e. Por exemplo, tendo definido $\rho_e = 0.7$, se a posição da chave gerada é menor ou igual a 0.7, então a solução descendente vai herdar o gene da solução elite. Caso contrário, herda da solução não elite. Uma ilustração deste processo encontra-se na figura 4.2, considerando um cromossoma de tamanho 4.

Solução elite	0,64	0,57	0,78	0,44
Solução não elite	0,47	0,52	0,63	0,77
Chaves aleatórias $\rho_e = 0{,}59$	0,34	0,52	0,86	0,41
Solução descendente	0,64	0,57	0,63	0,44

Figura 4.2: Operação de cruzamento uniforme parametrizado

Tal como indica a figura 4.2, é feita uma comparação *gene a gene* da chave gerada aleatoriamente e o valor considerado de ρ_e. A solução descendente herdou da solução elite os genes 1, 2 e 4 e da solução não elite o gene 3.

Quando a nova geração está completa, isto é, a população *k+1* tem *p* soluções, é verificado o *fitness* das mesmas e, posteriormente, são divididas em soluções elite e não elite, dependendo do seu valor.

O *decoder*, referido anteriormente, é a ferramenta que permite descodificar o cromossoma de modo a poder extrair a informação correspondente ao mapa de exames, para o problema da dissertação. O BRKGA tem como objetivo ser utilizado como uma *framework* e por isso é constituído por partes independentes e dependentes ao problema. As partes independentes não têm conhecimento do problema em questão, e.g. processos de cruzamento e mutação, e as partes dependentes permitem ligar o problema à meta-heurística. É nesta última que o *decoder* se insere. Desta forma, para poder desenvolver um BRKGA é apenas necessário definir a representação do cromossoma e o seu *decoder*. Informações mais detalhadas acerca da *framework* são fornecidas na descrição do sistema de apoio à decisão 6.

Concluindo, é possível identificar todos os passos do algoritmo e estabelecer uma sequência dos mesmos, como é representado na figura 4.3.

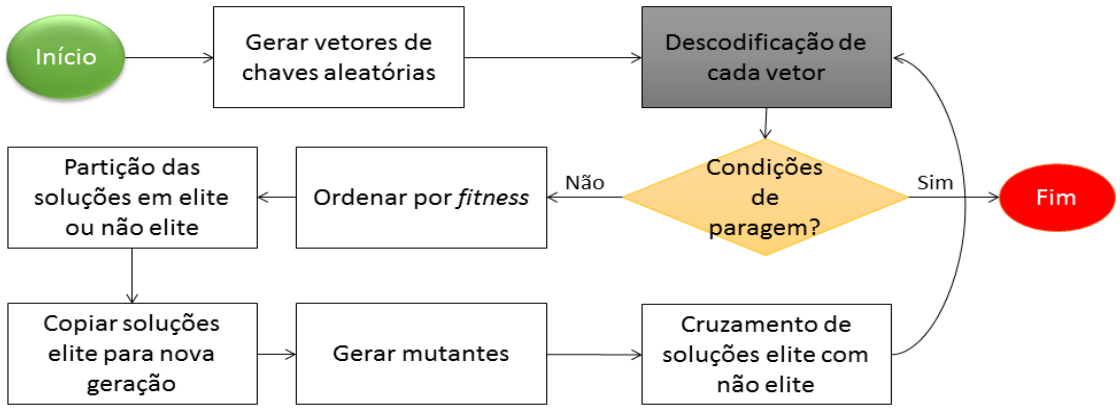


Figura 4.3: Flowchart da sequência do algoritmo

4.2.2 Reinicialização do BRKGA

A maioria das meta heurísticas que utilizam métodos estocásticos de pesquisa têm uma variável de avaliação comum que é o tempo à solução ótima. Esta variável tem um distribuição acumulada empírica[5] que assemelha a uma exponencial com deslocamento [34].

Foram efetuados vários estudos à volta deste tópico, conduzidos por Resende et al., em [35], num problema de recobrimento de triplas de Steiner ou Aiex et al., em [36]. Nesta medida, Resende efetuou testes sobre o BRKGA executando 100 vezes o algoritmo, cada vez usando uma semente diferente para o gerador de números aleatórios, observando o número de iterações que o algoritmo levou para encontrar a solução ótima. Segue a figura 4.4 extraída de [34].

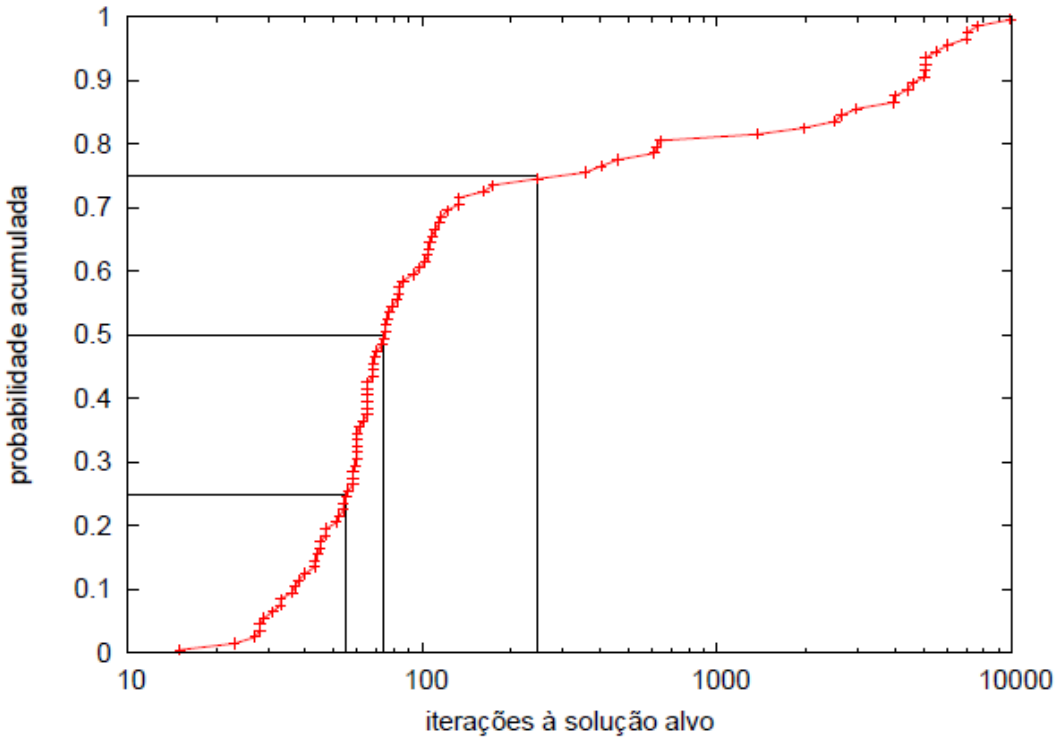


Figura 4.4: Distribuição das iterações a uma solução alvo/ótima do BRKGA sem reinicialização

De acordo com os resultados obtidos, 25% das experiências conduzidas encontram a solução ótima em menos de 55 iterações; 50% demoraram no máximo 74 iterações e 75% no máximo 245. Em adição, 10% precisaram mais de 4597 iterações, 5% de 5532 iterações e 2% de 7061, sendo que a execução mais demorada foi de 9903 iterações.

Através da figura é possível observar que a probabilidade de o algoritmo necessitar mais de 246 iterações é de 25%. Resende decidiu comparar um BRKGA sem reinicialização com outro que reinicializa a cada 246 iterações, sem que a melhor solução encontrada tenha sido substituída.

[5] Gráfico que representa cada valor por percentagem acumulada da amostra que são iguais ou menores a esse valor, conectando cada ponto numa linha escalonada.

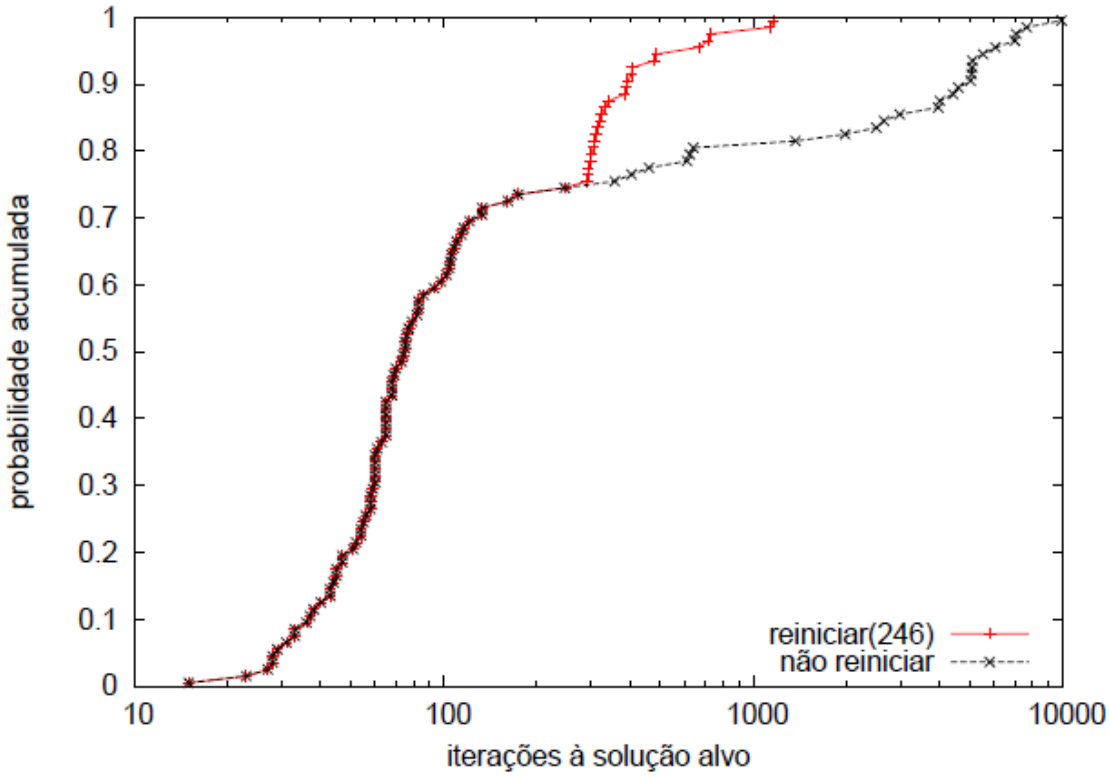


Figura 4.5: Distribuição das iterações a uma solução alvo/ótima do BRKGA com reinicialização e sem, no problema de recobrimento de triplas de Steiner.

Esta experiência, ilustrada na figura 4.5, levou a concluir que o número máximo de iterações que o algoritmo demora a encontrar a melhor solução é menor quando se reinicia a cada 246 iterações.

4.3 Aplicações

Ultrapssando os pormenores técnicos e introduzindo os usos práticos do algoritmo, a sua aplicação atingiu vários setores. Em 2005 [37], Gonçalves, Mendes e Resende propuseram um BRKGA para o problema de *job-shop scheduling* com o objetivo de minimizar o *makespan*. Este algoritmo foi comparado com diversas meta-heurísticas estado de arte, produzindo as melhores soluções em 72% dos problemas-teste.

Em 2007 [38], Gonçalves propôs um BRKGA para o problema de empacotamento bidimensional onde, igualmente, comparado a várias meta-heurísticas da literatura, foi capaz de produzir melhores soluções que a concorrência.

Entre outras aplicações, na área das telecomunicações, problemas de redes rodoviárias, entre outros, o BRKGA foi sempre capaz de superar os resultados recordistas de outras heurísticas.

Capítulo 5

Modelo proposto para o problema de calendarização de exames

O modelo proposto para o problema de calendarização de exames foi elaborado tendo como bases os diferentes modelos expostos na literatura. Foi igualmente desenvolvido respeitando as considerações expressas na Descrição do Problema, capítulo 3, assim como as contribuições da direção do Mestrado em Engenharia Eletrotécnica de Computadores. Apresenta-se numa primeira etapa o modelo matemático do problema idealizado, seguindo-se a abordagem às diversas restrições em contexto de programação, com introdução da estrutura do cromossoma e os diversos parâmetros configurados.

5.1 Modelo Matemático

5.1.1 Índices e Conjuntos

- $\mathscr{E}$ é o conjunto de todos os exames $e : e \in \mathscr{E} = \{1,\ldots,E\}$;
- $\mathscr{E}_N$ é o conjunto de todos os exames que se realizam na época normal;
- $\mathscr{E}_R$ é o conjunto de todos os exames que se realizam na época de recurso;
- $\mathscr{E} = \mathscr{E}_N \bigcup \mathscr{E}_R$, todos os exames se realizam na época normal ou na época de recurso;
- $\mathscr{T}$ é o conjunto de todos os períodos $t : t \in \mathscr{T} = \{1,\ldots,T\}$;
- $\mathscr{D}$ é o conjunto de *clusters* de períodos $\mathscr{D} = \{D_1, D_2, \ldots, D_D\}$ tal que:
  - os *clusters* não têm períodos em comum, $\bigcap_{d=1}^{D} D_d = \emptyset$
  - todos os períodos pertencem ao conjunto de períodos $\mathscr{T}$, $\forall d \in \{1,\ldots,D\}, D_d \subset \mathscr{T}$;
  - a reunião dos *clusters* é o conjunto $\mathscr{T}$, $\bigcup_{d=1}^{D} D_d = \mathscr{T}$;

- $\mathscr{M} = \{CP_1, CP_2, \ldots, CP_M\}$, em que cada subconjunto representa uma combinação possível de exames[1], $\mathscr{M}_i \subset \mathscr{E}$;

5.1.2 Dados do problema

- Número de estudantes em comum entre cada par de exames e e j, representado por uma matriz simétrica X com elementos x_{ej}.
- Anos curriculares da unidade curricular correspondente ao exame e.

  A partir desses dados podem-se construir as seguintes matrizes binárias:

  - Y com elementos $y_{ej} = 1$ quando o par de exames e e j corresponde a unidades curriculares do mesmo ano e $= 0$ caso contrário;
  - Z com elemento $z_{ej} = 1$ quando o par de exames e e j corresponde a unidades curriculares de anos letivos consecutivos e $= 0$ caso contrário.
- Conjuntos de *clusters* que representam, no ponto de vista do estudante, a combinação de exames que cada aluno tem em cada ano curricular, para cada ramo do curso, $\mathscr{CL} = \{C_1, C_2, \ldots\}, \quad C \subset \mathscr{E}$;
- Número mínimo e máximo de períodos entre exames:
  - na época normal $dist_{N_{MIN}}$, $dist_{N_{MAX}}$;
  - na época de recurso $dist_{R_{MIN}}$ e $dist_{R_{MAX}}$.
- Número mínimo e máximo de períodos entre épocas de exame, $dist_{NR_{MIN}}$ e $dist_{NR_{MAX}}$;
- Salas: capacidade para exame e caraterização (sala normal ou sala de computadores);

5.1.3 Variáveis de decisão

Este problema apenas requer um tipo de variáveis de decisão binárias que indicam se um exame e ocorre ou não no período t:

$$\delta_{et} = \begin{cases} 1, \text{ se exame } e \text{ ocorre no período } t \\ 0, \text{ se não} \end{cases}$$

5.1.4 Restrições Fortes

As restrições fortes têm de ser cumpridas para que uma solução seja admissível para o problema. O conjunto das restrições fortes do problema é apresentado a seguir.

[1] Estes conjuntos são denominados de possíveis pois, não tendo acesso à informação "um conjunto de alunos que frequenta a unidade curricular X, frequenta as unidades Y, Z;outro conjunto frequenta Z,M."É assumido o pior caso possível em que um estudante que frequente a UC X também frequente todas as UCs em comum com X.

Não podem ocorrer mais do que 3 exames em cada período:

$$\forall t \in \mathscr{T} \quad \sum_{e \in \mathscr{E}} \delta_{et} \leq 3; \tag{5.1}$$

Exames que ocorrem no mesmo período não podem ter estudantes em comum:

$$\forall t \in \mathscr{T} \quad \sum_{e=1}^{E-1} \sum_{j=e+1}^{E} \delta_{et} \cdot \delta_{jt} \cdot x_{ej} = 0; \tag{5.2}$$

Exames que ocorrem em períodos adjacentes não podem ter estudantes em comum:

$$\forall t \in \{1,\ldots,T-1\} \quad \sum_{e=1}^{E-1} \sum_{j=e+1}^{E} \delta_{e,t} \cdot \delta_{j,t+1} \cdot x_{ej} = 0; \tag{5.3}$$

Exames do mesmo ano não podem ocorrer no mesmo dia:

$$\forall D_d \in \mathscr{D} \quad \forall e \in \{1,\ldots,E-1\} \quad \forall j \in \{e+1,\ldots,E\} \quad \prod_{t \in D_d} \delta_{et} \cdot \delta_{jt} \cdot y_{ej} = 0; \tag{5.4}$$

Exames de anos consecutivos não podem ocorrer no mesmo dia:

$$\forall D_d \in \mathscr{D} \quad \forall e \in \{1,\ldots,E-1\} \quad \forall j \in \{e+1,\ldots,E\} \quad \prod_{t \in D_d} \delta_{et} \cdot \delta_{jt} \cdot z_{ej} = 0; \tag{5.5}$$

O primeiro exame da época de recurso ocorre após o último exame da época normal:

$$\max_{t \in \mathscr{T}} (\delta_{e \in \mathscr{E}_N, t} \cdot t) < \min_{t \in \mathscr{T}} (\delta_{j \in \mathscr{E}_R, t} \cdot t). \tag{5.6}$$

Garantir que apenas ocorrem dois exames para cada unidade curricular, um na época normal e outro na época de recurso:

$$\forall e \in \mathscr{E}_N \quad \sum_{t \in \mathscr{T}} \delta_{et} = 1; \tag{5.7}$$

$$\forall e \in \mathscr{E}_R \quad \sum_{t \in \mathscr{T}} \delta_{et} = 1; \tag{5.8}$$

5.1.5 Restrições Fracas

As restrições fracas podem não ser cumpridas numa solução admissível do problema, mas o seu não cumprimento é penalizado na função objetivo.

Exames com estudantes em comum não podem ocorrer no mesmo dia:

$$\forall D_d \in \mathscr{D} \quad \forall e \in \{1,\ldots,E-1\} \quad \forall j \in \{e+1,\ldots,E\} \quad \prod_{t \in D_d} \delta_{et} \cdot \delta_{jt} \cdot x_{ej} = 0; \tag{5.9}$$

O número de períodos de tempo entre a época normal e a época de recurso deve estar entre os respetivos limites $dist_{NR_{MIN}}$ e $dist_{NR_{MAX}}$:

$$dist_{NR_{MIN}} \leq \min_{t\in\mathscr{T}}(\delta_{j\in\mathscr{E}_R,t}\cdot t) - \max_{t\in\mathscr{T}}(\delta_{e\in\mathscr{E}_N,t}\cdot t) \leq dist_{NR_{MAX}}. \tag{5.10}$$

Para exames pertancentes ao mesmo *cluster* o número de períodos de tempo entre exames na época normal e na época de recurso deve estar entre os respetivos limites:

$$\forall C\in\mathscr{CL} \quad \forall i=1,...,dim(C)-1 \quad dist_{N_{MIN}} \leq \left|\sum_{t\in\mathscr{T}}\delta_{C_{i_N},t}\cdot t - \sum_{t\in\mathscr{T}}\delta_{C_{i_N+1},t}\cdot t\right| \leq dist_{N_{MAX}}; \tag{5.11}$$

$$\forall C\in\mathscr{CL} \quad \forall i=1,...,dim(C)-1 \quad dist_{R_{MIN}} \leq \left|\sum_{t\in\mathscr{T}}\delta_{C_{i_N},t}\cdot t - \sum_{t\in\mathscr{T}}\delta_{C_{i_N+1},t}\cdot t\right| \leq dist_{R_{MAX}}; \tag{5.12}$$

5.1.6 Função objetivo

$$\begin{aligned}
min: \quad & \sum_{j=1}^{dim(\mathscr{CL})}\sum_{i=1}^{dim(\mathscr{CL}_j)-1} \frac{\left|\left|\sum_{t\in\mathscr{T}}\delta_{\mathscr{CL}_{ji_N},t}\cdot t - \sum_{t\in\mathscr{T}}\delta_{\mathscr{CL}_{j,i_N+1},t}\cdot t\right| - dist_{N_{MAX}} + R\right|}{\left|\sum_{t\in\mathscr{T}}\delta_{\mathscr{CL}_{ji_N},t}\cdot t - \sum_{t=1}^{T}\delta_{\mathscr{CL}_{j,i_N+1},t}\cdot t\right| - dist_{N_{MAX}} - R} + \\
& \sum_{j=1}^{dim(\mathscr{CL})}\sum_{i=1}^{dim(\mathscr{CL}_j)-1} \frac{\left|\sum_{t\in\mathscr{T}}\delta_{\mathscr{CL}_{ji_N},t}\cdot t - \sum_{t\in\mathscr{T}}\delta_{\mathscr{CL}_{j,i_N+1},t}\cdot t\right| - dist_{N_{MIN}} + R}{-\left|\left|\sum_{t\in\mathscr{T}}\delta_{\mathscr{CL}_{ji_N},t}\cdot t - \sum_{t\in\mathscr{T}}\delta_{\mathscr{CL}_{j,i_N+1},t}\cdot t\right| - dist_{N_{MIN}} - R\right|} + \\
& \sum_{j=1}^{dim(\mathscr{CL})}\sum_{i=1}^{dim(\mathscr{CL}_j)-1} \frac{\left|\left|\sum_{t\in\mathscr{T}}\delta_{\mathscr{CL}_{j,i_R},t}\cdot t - \sum_{t\in\mathscr{T}}\delta_{\mathscr{CL}_{j,i_R+1},t}\cdot t\right| - dist_{R_{MAX}} + R\right|}{\left|\sum_{t\in\mathscr{T}}\delta_{\mathscr{CL}_{ji_R},t}\cdot t - \sum_{t\in\mathscr{T}}\delta_{\mathscr{CL}_{j,i_R+1},t}\cdot t\right| - dist_{R_{MAX}} - R} + \\
& \sum_{j=1}^{dim(\mathscr{CL})}\sum_{i=1}^{dim(\mathscr{CL}_j)-1} \frac{\left|\sum_{t\in\mathscr{T}}\delta_{\mathscr{CL}_{ji_R},t}\cdot t - \sum_{t\in\mathscr{T}}\delta_{\mathscr{CL}_{j,i_R+1},t}\cdot t\right| - dist_{R_{MIN}} + R}{-\left|\left|\sum_{t\in\mathscr{T}}\delta_{\mathscr{CL}_{ji_R},t}\cdot t - \sum_{t\in\mathscr{T}}\delta_{\mathscr{CL}_{j,i_R+1},t}\cdot t\right| - dist_{R_{MIN}} - R\right|} + \\
& \sum_{j=1}^{dim(\mathscr{M})}\sum_{i=1}^{dim(\mathscr{M}_j)} \frac{\min_{t\in\mathscr{T}}(\delta_{\mathscr{M}_{ji_R},t}\cdot t) - \max_{t\in\mathscr{T}}(\delta_{\mathscr{M}_{ji_N},t}\cdot t) - dist_{NR_{MIN}} + R}{-\left|\min_{t\in\mathscr{T}}(\delta_{\mathscr{M}_{ji_R},t}\cdot t) - \max_{t\in\mathscr{T}}(\delta_{\mathscr{M}_{ji_N},t}\cdot t) - dist_{NR_{MIN}} - R\right|} + \\
& \sum_{j=1}^{dim(\mathscr{M})}\sum_{i=1}^{dim(\mathscr{M}_j)} \frac{\left|\min_{t\in\mathscr{T}}(\delta_{\mathscr{M}_{ji_R},t}\cdot t) - \max_{t\in\mathscr{T}}(\delta_{\mathscr{M}_{ji_N},t}\cdot t) - dist_{NR_{MAX}} + R\right|}{\min_{t\in\mathscr{T}}(\delta_{\mathscr{M}_{ji_R},t}\cdot t) - \max_{t\in\mathscr{T}}(\delta_{\mathscr{M}_{ji_N},t}\cdot t) - dist_{NR_{MAX}} - R} + \\
& \sum_{i=1}^{dim(\mathscr{D})}\sum_{t=1}^{dim(\mathscr{D}_i)}\sum_{e=1}^{E-1}\sum_{j=e+1}^{E} \frac{\delta_{e,\mathscr{D}_{it}}\cdot\delta_{j,\mathscr{D}_{i,t+2}}\cdot x_{ej} + R}{\delta_{e,\mathscr{D}_{it}}\cdot\delta_{j,\mathscr{D}_{i,t+2}}\cdot x_{ej} - R},
\end{aligned} \tag{5.13}$$

onde R, representa um infinitésimo.

As quatro primeiras parcelas da soma da função objetivo são relativas à restrição descrita através das equações 5.11 e 5.12, as duas primeiras são relativas à época normal, limite máximo e mínima, e as duas seguintes à época de recurso, limite máximo e mínimo. Sendo um problema de minimização, a cada verificação válida[2] é necessário subtrair um valor à função objetivo e aquando uma verificação inválida é adicionado um valor à mesma, sendo este valor -1 e 1, respetivamente.

Observando o caso da época normal de exames, as fórmulas para verificação dos limites máximo e mínimo são diferentes, tomando as seguintes simplificações: $\frac{|\Delta - dist_{N_{MAX}} + R|}{\Delta - dist_{N_{MAX}} - R}$ e $\frac{\Delta - dist_{N_{MIN}} + R}{-|\Delta - dist_{N_{MIN}} - R|}$, para os limites máximo e mínimo, respetivamente, onde Δ representa o intervalo absoluto entre dois exames pertencentes no mesmo *cluster*. A comparação deste intervalo com os limites de especificação (p.e. $\Delta - dist_{N_{MAX}}$) pode ora ser um valor negativo, provando-se ser inferior ao limite; ser 0, provando-se estar na fronteira do limite de especificação ou ser um valor positivo, provando-se estar acima do limite de especificação.

No caso particular de comparação com o limite máximo, recordando tratar-se de um problema de minimização, aquando o cumprimento da restrição deve ser subtraído o valor -1 à função objetivo o que irá ocorrer quando a comparação é inferior ao limite ou está na fronteira do limite de especificação. Assim, através da expressão acima referida, é possível assegurar que quando $\Delta - dist_{N_{MAX}}$ é ≤ 0 a função objetivo é subtraída corretamente. Seguem as seguintes verificações:

- $\Delta - dist_{N_{MAX}} < 0$

$$\frac{|-M+R|}{-M-R} = \frac{M-R}{-M-R} \approx -1 \tag{5.14}$$

- $\Delta - dist_{N_{MAX}} = 0$

$$\frac{|0+R|}{0-R} = \frac{R}{-R} = -1 \tag{5.15}$$

- $\Delta - dist_{N_{MAX}} > 0$

$$\frac{|M+R|}{M-R} = \frac{M+R}{M-R} \approx 1 \tag{5.16}$$

O mesmo acontece no caso de comparação com o limite mínimo, a cada cumprimento da restrição é subtraído o valor -1 à função objetivo que acontece quando a comparação é superior ao limite ou na fronteira do limite de especificação. Assim, através da segunda expressão acima referida é possível assegurar que quando $\Delta - dist_{N_{MIN}}$ é ≥ 0 a função objetivo é subtraída corretamente. Seguem as seguintes verificações:

- $\Delta - dist_{N_{MIN}} > 0$

$$\frac{M+R}{-|M-R|} = \frac{M+R}{-M+R} \approx -1 \tag{5.17}$$

- $\Delta - dist_{N_{MIN}} = 0$

$$\frac{0+R}{-|0-R|} = \frac{R}{-R} = -1 \tag{5.18}$$

[2]Define-se por verificação válida sempre que cada observação das restrições é cumprida.

- $\Delta - dist_{N_{MIN}} < 0$

$$\frac{-M+R}{-|-M-R|} = \frac{-M+R}{-M-R} \approx 1 \tag{5.19}$$

O mesmo se aplica para a quinta e sexta parcela da soma da função objetivo, referentes à restrição "O número de períodos de tempo entre a época normal e a época de recurso deve estar entre os respetivos limites", onde se faz a comparação com os limites mínimo e máximo.

Quanto à última parcela, referente à restrição "Exames com estudantes em comum não podem ocorrer no mesmo dia", considera-se que foi cumprida quando $\delta_{e,\mathscr{D}_{it}} \cdot \delta_{j,\mathscr{D}_{i,t+2}}$ toma o valor 0, ou seja, não há alunos em comum nos exames marcadas para o mesmo dia. Sempre que esta verificação é feita, decrementa-se a função objetivo de -1, caso contrário incrementa-se a mesma com valor 1.

5.2 Abordagem ao problema na perspetiva computacional

Nesta secção é apresentada a forma como as diferentes restrições acima apresentadas foram interpretadas em termos do BRKGA e traduzidas para linguagem algorítmica e implementadas no descodificador da ferramenta. Os assuntos têm a seguinte ordem: estrutura do cromossoma, função objetivo, abordagem às restrições fortes e abordagem às restrições fracas, complementadas estas últimas por trechos de códigos da *framework*.

5.2.1 Estrutura do cromossoma

Relembrando o foco da dissertação, no final espera-se obter uma solução que traduza um mapa de exames, mais especificamente, cada exame terá uma data alocada para acontecer. Esta solução é obtida através da estrutura do cromossoma, e a sua estruturação é da maior importância para poder entender e compreender a informação que acarreta. Tendo em conta que um cromossoma é constituído por genes e esses genes possuem informação, decidiu-se que a estrutura do cromossoma associe a cada gene um exame e a informação contida nos genes seja a data a que o exame está alocado.

Se a estrutura fosse a inversa, genes correspondentes aos vários períodos do calendário e valor dos genes correspondente a cada exame, dificultaria a imposição de todos os exames terem de ocorrer exata e obrigatoriamente duas vezes.

Dado isto, foi ainda decidido que cada par de genes consecutivos corresponderia a uma única unidade curricular, sendo que o primeiro corresponde ao exame de época normal e o segundo ao exame de época de recurso. Desta forma é possível certificar que cada unidade curricular tem dois exames com data bem definidos e com acesso direto a qualquer época de exames de cada unidade curricular.

Como ilustra a figura 5.1, cada par de genes consecutivos é referente a um único exame, em que o primeiro gene representa o exame de época normal e o segundo de época de recurso.

O cromossoma não é nada mais que um vetor, de dimensão igual ao número de dias do calendário de exames, e a partir deste é criado um vetor *time*, em que cada posição corresponde a

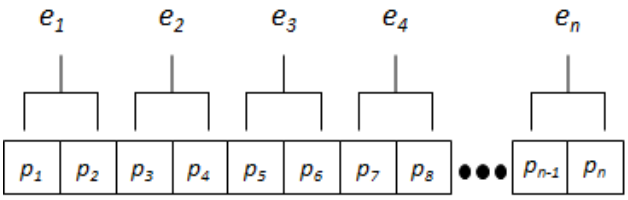


Figura 5.1: Estrutura do cromossoma

um período do calendário e em cada período há um vetor com os índices dos exames associados. Tendo estes dois vetores presentes, a fase de descodificação torna-se mais evidente, devido à facilidade de resposta às perguntas: "Em que período está o exame 2?"ou "Que exames estão no período 20?". Esta mecânica nasceu a partir da ideia de Čupić, Golub e Jakobović apresentada no capítulo 2, secção 2.1.5, que embora tenha uma lógica diferente, o objetivo é o mesmo.

5.2.2 Função objetivo

Relembrando os objetivos indicados no capítulo 3, no ponto de vista "estudante"o maior objetivo será ter os exames bem espaçados entre si. Contudo é necessário realçar qual a utilidade desta ferramenta. Esta serve para fornecer um apoio à decisão na elaboração de um mapa de exames, não fornecer o melhor mapa, ou o mais espaçado, mas sim um mapa que cumpre as restrições fundamentais que um calendário tem de ter. A tarefa manual de verificar a independência entre unidades curriculares é bastante morisa, o que incita à criação de uma ferramenta como esta.

Ora, no ponto de vista "geral", o objetivo será evitar quaisquer conflitos, em todo o calendário elaborado, sejam de sobreposição, de congestionamento, ou outros. Só depois desses critérios cumpridos é que se dá lugar ao espaçamento entre exames/conjuntos de exames. Dito isto, a função objetivo, ou *fitness*, está diretamente correlacionada com as restrições fortes e fracas do problema, tratando-se de uma combinação linear das mesmas.

Inicialmente foram consideras duas formas de definir a função objetivo:

1. através da combinação linear das restrições fortes e fracas;
2. através da combinação linear das restrições fortes ou fracas.

No caso 1, a função objetivo é dada pela soma dos custos associados às restrições fortes e fracas. Por outro lado, o caso 2 diferencia-se do 1 na medida em que apenas usa os custos das restrições fortes, ou das restrições fracas, nunca ambos em simultâneo. Este segundo método é constituído por duas fases, numa primeira são analisadas as restrições fortes e verifica-se se são cumpridas (custos = 0). Se não for o caso a função objetivo toma como valores a combinação linear das restrições fortes. Caso os custos sejam nulos, são analisadas as restrições fracas e a função objetivo passa a ser uma combinação linear das restrições fracas.

Após testes iniciais da função objetivo, e posteriormente suportado pelos testes finais, decidiu-se optar pelo método 2, devido à incapacidade do método 1 em encontrar uma solução admissível,

isto é, uma solução em que as restrições fortes são cumpridas na sua totalidade. Assim, as restrições fracas apenas são contabilizadas se as restrições fortes forem satisfeitas, pois se não fosse o caso a contabilização das anteriores seria completamente inútil. Os benefícios extraídos são a diminuição do tempo de execução da iteração e o aumento do foco do algoritmo para o cumprimento das restrições fortes, além da obtenção de uma solução admissível.

Matematicamente, a função objetivo é a seguinte:

$$fitness = w_1 \cdot y(fortes) + w_2 \cdot \delta \cdot z(fracas), \tag{5.20}$$

onde w_1 e w_2 representam os pesos atribuídos às restrições fortes e fracas, respetivamente e δ uma variável booleana que toma o valor de 1 se as restrições fortes forem cumpridas, ou 0 se não; y(fortes) e z(fracas) correspondem às combinações lineares dos custos/penalizações das restrições fortes e fracas, respetivamente.

5.2.3 Abordagem às restrições fortes

1. Não podem ocorrer mais que 3 exames no mesmo período: tal como a descrição indica, procura-se penalizar o facto de ocorrer sobrecargas nos diferentes períodos, procurando distribuir os exames ao longo do calendário. Sendo o vetor *time* um vetor de vetores, em que estes últimos correspondem aos exames em cada período do calendário. A restrição em linguagem máquina encontra-se no anexo A, na secção A.3, nas linhas 68 a 75.

   (a) Aceder a cada posição do vetor *time* e verificar se o tamanho do vetor nessa posição é superior a 3;

   (b) Caso se verifique incrementa um contador;

   (c) Depois de verificado o vetor *time* é verificado se o contador é diferente de 0, se sim incrementa um custo [3] de 999.

2. Exames pertencentes ao mesmo período não podem ter estudantes em comum. A restrição em linguagem máquina encontra-se no anexo A, na secção A.3, nas linhas 82 a 107.

   (a) Aceder a cada posição do vetor *time* extraindo os índices dos exames presentes;

   (b) Analisar qual o conflito entre todos os exames extraído, caso conflito for superior a 0 então é incrementado um custo com o número de estudantes em conflito.

3. Exames em períodos adjacentes não podem ter estudantes em comum: tal como é indicado, nenhum estudante pode sair de um exame e ter um novo exame, no período seguinte, no mesmo dia. A restrição em linguagem máquina encontra-se no anexo A, na secção A.3, nas linhas 115 a 185.

   (a) Aceder de 3 em 3 posições consecutivas do vetor *time*;

[3] Cada custo anunciado nas abordagens às restrições são independentes entre si

(b) Os exames são copiados para 3 vetores, em que o 1º corresponde ao 1º período, o 2º ao 2º e o 3º ao 3º período;

(c) É feita uma comparação em pares entre o vetor nº2 e os restantes;

(d) Caso exista conflito é incrementado um custo equivalente ao número de estudantes em conflito;

(e) Caso o conflito for entre exames da mesma unidade curricular, incrementa um custo igual a 100.

4. Exames do mesmo ano curricular não podem ocorrer no mesmo dia: esta restrição está em sinergia com a restrição 3, primeiro devido a ser elaborada uma análise diária e em segundo porque a restrição 3 já faz 2/3 do trabalho através da verificação dos períodos adjacentes num dia. Isto porque os grupos exames do mesmo ano curricular estão organizados em *clusters* e ocorrem sempre em conjunto. A restrição em linguagem máquina encontra-se no anexo A, na secção A.3, nas linhas 166 a 171.

   Dado isto é apenas verificado a relação entre o primeiro período do dia e o último:

   (a) Aceder de 3 em 3 posições consecutivas do vetor *time*;

   (b) Os exames são copiados para 3 vetores, em que o 1º corresponde ao 1º período, o 2º ao 2º e o 3º ao 3º período;

   (c) É feita uma comparação entre o 1º e 3º vetor;

   (d) Antes de iniciar a comparação é chamada uma função para verificar se o par de exames a ser comparado pertence ao mesmo *cluster*;

   (e) Caso pertença é incrementada a função objetivo com um custo de 100;

   (f) Caso contrário, verifica-se o número de estudantes em conflito e incrementa-se um custo associado à restrição fraca associada à equação 5.9, explicado mais à frente.

5. Exames de anos consecutivos não podem ocorrer no mesmo dia: são contabilizados os primeiros três anos curriculares, esta restrição está embutida em 3 visto ser feita uma análise ao dia inteiro. A restrição em linguagem máquina encontra-se no anexo A, na secção A.3, nas linhas 127 a 140 e 178 a 182.

   (a) Aceder de 3 em 3 posições consecutivas do vetor *time*;

   (b) À medida que cada exame é copiado para os vetores em 3, é executada uma função que vai indicar a que ano pertence e é incrementado um contador associado ao ano curricular em questão;

   (c) Depois de todos os exames copiados são verificados os contadores, caso o contador associado ao 1º e 2º ano forem superiores a 0 ou os contadores de 2º e 3º ano forem superiores a 0, é incrementado um custo à função objetivo de 999.

6. Garantir que, para qualquer calendário possível, o primeiro exame de época de recurso inicia-se posteriormente ao último exame de época normal: como indica, cada estudante, cada um com um calendário associado, apenas pode iniciar a época de recurso após a conclusão da época normal[4]. A restrição em linguagem máquina encontra-se no anexo A, na secção A.3, nas linhas 191 a 224.

   (a) É analisado para cada unidade curricular, quais são as unidades curriculares que têm alunos em comum;

   (b) As que tiverem alunos em comum, é copiado o período do exame normal, e o seu índice, para um vetor apropriado e o do exame de recurso para outro vetor apropriado;

   (c) Os vetores são então ordenados crescentemente de acordo com os valores dos períodos;

   (d) É identificado o maior período do vetor de época normal;

   (e) Para cada posição do vetor correspondente aos exames de recurso, é verificado se o período é inferior ao valor máximo definido anteriormente;

   (f) Se se verificar, o período é copiado para um vetor auxiliar;

   (g) É agora comparado o vetor de época normal com o vetor auxiliar, se um período do primeiro vetor for superior ao período do vetor auxiliar e se ambos pertencerem a unidades curriculares diferentes, é incrementado um custo com valor igual ao número de estudantes em comum entre as duas unidades curriculares;

   (h) Caso pertencerem à mesma unidade curricular é incrementado um custo igual a 200.

5.2.4 Abordagem às restrições fracas

1. Cada estudante apenas tem 1 exame por dia: esta restrição opera em conjunto com as restrição forte 4, que por sua vez se encontra associada à restrição forte 3. Quando na restrição 4 é verificado se o par de exames pertence ao mesmo *cluster*, como introduzido na abordagem, é acionada esta restrição. A restrição em linguagem máquina encontra-se no anexo A, na secção A.3, nas linhas 166 a 169.

   (a) Aceder de 3 em 3 posições consecutivas do vetor *time*;

   (b) Os exames são copiados para 3 vetores, em que o 1º corresponde ao 1º período, o 2º ao 2º e o 3º ao 3º período;

   (c) É feita uma comparação entre o 1º e 3º vetor;

   (d) Antes de iniciar a comparação é chamada uma função para verificar se o par de exames a ser comparado pertence ao mesmo *cluster*;

   (e) Caso não pertença é incrementada a função objetivo com um custo igual ao número de estudantes em conflito do par de exames.

[4]Não existe "linha de água"que separa a época normal da época de recurso devido à quantidade de exames que existem no curso, ao invés, a linha é então homogénea e pode ser diferente em qualquer calendário

2. Maximizar o intervalo de tempo entre a época normal e a época de recurso: o procedimento é similar ao ponto 6, na abordagem às restrições fortes. É necessário contabilizar todos os calendários possíveis para cada unidade curricular. A restrição em linguagem máquina encontra-se no anexo A, na secção A.3, nas linhas 266 a 284.

   (a) É analisado para cada unidade curricular, quais são as restantes unidades curriculares que têm alunos em comum;

   (b) Os períodos designados para essas unidades curriculares, quer para época normal como para recurso, são copiados para um vetor apropriado;

   (c) Aquando da cópia dos períodos é feito um ajuste aos valores de modo a contabilizar os fins de semana;

   (d) Os vetores são ordenados por ordem crescente;

   (e) É calculada a diferença entre o mínimo período do vetor de época de recurso com o máximo período do vetor de época normal;

   (f) Se essa diferença estiver fora dos limites considerados aceitáveis é incrementado um custo à função objetivo de 100;

   (g) Caso a diferença se encontre dentro dos limites é decrementado um valor de 100 à função objetivo.

3. Maximizar o intervalo de tempo entre exames (época normal e recurso) pertencentes ao mesmo *cluster*: tal como a definição indica, para cada *cluster* existente é calculado o intervalo de tempo entre exames em qualquer época de exames e interpretado segundo limites de controlo. A restrição em linguagem máquina encontra-se no anexo A, na secção A.3, nas linhas 226 a 264 .

   (a) Ciclicamente são importados os *clusters* de exames;

   (b) Os períodos dos exames de época normal e época de recurso são copiados para vetores apropriados;

   (c) Aquando da cópia dos períodos é feito um ajuste aos valores de modo a contabilizar os fins de semana;

   (d) Os vetores são ordenados por ordem crescente de período;

   (e) Cada vetor é analisado comparando períodos consecutivos, calculando o intervalo de tempo verificando se se encontra fora dos limites aceitáveis;

   (f) Caso se encontre fora dos limites é incrementado um valor de custo à função objetivo igual a 100;

   (g) Caso contrário é decrementado um valor de 100 à função objetivo.

5.2.5 Alocação nas salas

Como funcionalidade extra do programa, não incluído nos requisitos iniciais, e no intuito de lançar investigação futura relacionada com a alocação espacial de calendário de eventos, foi implementada uma rotina que, depois de executado o algoritmo BRKGA e extraída a melhor solução, a cada dia e a cada período do dia aloca uma sala a cada exame.

Em cada período, os exames pertencentes a esse período são ordenados num vetor por ordem decrescente de número de estudantes que vão participar. Do mesmo modo um vetor de salas é ordenado decrescente segunda as suas capacidades. Num ciclo, é percorrido o vetor de exames e as salas vão ser atribuídas a cada exame até o somatório das capacidades das salas atribuídas seja superior, ou igual, ao número de estudantes que vão atender ao exame em causa. Este ciclo é aplicado a cada período do calendário. Após a atribuição das salas a todos os exames é gerado um ficheiro *CSV* com os resultados.

O código de leitura dos dados e o algoritmo responsável por atribuir as salas aos exames em cada período está presente no anexo A, na secção A.4.

Capítulo 6

Descrição do Sistema de Apoio à Decisão

Neste capítulo irá descrever-se todos os constituintes do sistema de apoio à decisão, incluindo o algoritmo, a ferramenta de leitura de dados, os valores dos parâmetros, os pesos na função objetivo e os ficheiros de diagnóstico de teste.

6.1 Introdução à *framework*

Rodrigo F. Toso e Mauricio G.C. Resende, propõem uma biblioteca desenvolvida na linguagem C++ que traduz o algoritmo BRKGA numa interface de programação [39]. Com um design simples, o utilizador apenas necessita de implementar o descodificador do cromossoma e definir certos parâmetros e critérios de paragem do algoritmo.

Na interface do descodificador, *Decoder*, o utilizador pode escolher um de dois métodos a ser implementado:

- **double decode(std::vector<double>& chromosome) const:** permite ler e escrever num cromossoma;
- **double decode(const std::vector<double>& chromosome) const:** permite apenas leitura do cromossoma.

A diferença entre estes dois métodos reside na possibilidade de, no primeiro método, ser possível alterar os valores dos genes do cromossomas o que pode ser interessante quando aplicadas heurísticas de reconstrução. Para este caso, optou-se pelo segundo método, apenas leitura do cromossoma.

Nesta interface, tal como o nome indica, é onde o utilizador insere a sua interpretação dos genes do cromossoma, resultando no final na função objetivo. Um exemplo:

$$myFitness = \sum_{i=0}^{n-1} (i+1) \cdot chromosome[i]; \tag{6.1}$$

Os parâmetros que o utilizador deve introduzir, explicados em 4.2.1, são:

- n: número de genes de cada cromossoma[1];
- p: número de elementos em cada população;
- pe: fração da população constituída por elementos elite;
- pm: fração da população constituída por mutantes introduzidos durante o processo de evolução das gerações;
- rho: probabilidade de um cromossoma "filho"receber os genes do cromossoma "pai"elite;
- K: embora opcional, número de populações;
- MAXT: Número máximo de *threads* que podem ser utilizadas para efetuar descodificação em paralelo;
- X_INTVL: Períodos cíclicos em que as diferentes populações trocam soluções elite entre si;
- X_NUMBER: O número de soluções elite a serem trocados com outras populações;
- MAX_GENS: O número máximo de gerações a serem criadas;
- rngSeed: Semente que inicializa o gerador de números aleatórios;

Para além de definir estes parâmetros, o utilizador necessita ainda de definir a condição de paragem: (i) Executar até *n* gerações; (ii) Se valor da função objetivo for inferior a *x*, entre outros casos possíveis.

6.2 Leitura de dados

Para além do utilizador ter de implementar o descodificador e definir os valores dos parâmetros, é também necessário implementar um API para leitura de dados, quando os dados são fornecidos através de ficheiros. Tal como exposto em 5.1.2, os dados do problema consistem em:

- Matriz simétrica, *36x36*, que indica o número de estudantes em comum entre diversas disciplinas;
- Matriz, *14x36*, que representa as unidades curriculares que pertencem a cada um dos 14 *clusters* definidos;
- Nome de cada unidade curricular;

[1]Para o problema, *n* toma o valor 72 correspondente ao número de exames a serem marcados (época normal + recurso)

- Matriz que contém os índices das unidades curriculares que pertencem ao 1º, 2º e 3º ano curricular.

Os dados acima mencionados são organizados num ficheiro *.txt*, o ficheiro utilizado nas simulações encontra-se no anexo B, na secção B.2.

Como extra são também usados os dados necessários à marcação dos exames nas salas que são:

- Nome e capacidade das salas de exame escrito;
- Nome e capacidade das salas de exame em computador;
- Número de alunos inscritos a cada unidade curricular.

Como já referido no capítulo 5, secção 5.2.5, o código de leitura dos dados está presente no anexo A, na secção A.4, e os dados associados, no anexo B, secção B.3

6.3 Parâmetros - Parte independente

Os parâmetros mencionados na secção de introdução à ferramenta pertencem à parte independente do BRKGA, ou seja, são dados que não influenciam diretamente a função objetivo, mas ao serem utilizados nas operações de evolução das gerações são responsáveis, em parte, pela velocidade com que a função objetivo evolui.

Tabela 6.1: Valores *standard* dos parâmetros e a variação paramétrica para testes computacionais.

Parâmetros	Valor STD	Variação
p	100	100
pe	0.20	{0.10,0.15,0.20,0.25,0.30}
pm	0.15	{0.10,0.15,0.20,0.25,0.30}
rho	0.7	{0.6,0.7,0.8,0.9}
K	2	{1,2,3}
MAXT	2	2
X_INTVL	100	{50,60,70,80,90,100}
X_NUMBER	2	{1,2,3}
MAX_GENS	3000	3000

6.4 Parâmetros - Parte dependente

Quanto aos parâmetros dependentes do problema, alguns já introduzidos no modelo matemático, foram definidos valores que podem ser alterados conforme as especificações desejadas pelo utilizador, antes de inicializar o algoritmo.

Em adição às já mencionadas, surge como parâmetro o número de semanas em que o mapa de exames tem de ser construído e um valor de referência, *threshold*, que permite ao utilizador

escolher qual o número mínimo de alunos que o programa deve considerar ao verificar a restrição fraca correspondente ao número de exames que um estudante pode ter por dia. Por exemplo, se definirmos um valor de referência de 10 estudantes e num caso específico em que no período das 9h00 ocorre o exame de Análise Matemática I e no período das 17h00 ocorre Teoria do Controlo tendo estas unidades curriculares 9 estudantes em comum, o algoritmo irá desconsiderar essa situação, sem prejudicar a função objetivo. Durante a fase de validação do modelo o *threshold* considerado foi de 0, permitindo considerar todas e quaisquer situações.

Tabela 6.2: Valores *standard* dos parâmetros dependentes.

Parâmetros	Valor
semanas	5
$dist_{normal_{MIN}}$	11
$dist_{normal_{MAX}}$	15
$dist_{recurso_{MIN}}$	8
$dist_{recurso_{MAX}}$	11
$dist_{normal-recurso_{MIN}}$	8
$dist_{normal-recurso_{MAX}}$	11
threshold	0

Na fase de testes computacionais, estes parâmetros não foram variados pois foram considerados como críticos para a aceitação das soluções pelo decisor.

6.5 Critério de paragem

Como critério de paragem foi decidido estabelecer um limite máximo de gerações a serem geradas, valor apresentado na tabela 6.1. Este valor foi fixado a partir de testes preliminares onde se constou que cerca de 70% das execuções do algoritmo obtiveram a melhor solução, sem se alterar, a partir das 2000 gerações.

6.6 Pesos na função objetivo

Como referido no capítulo 5, na secção Função objetivo, o valor de cada solução é ora composto pela combinação linear das restrições fortes ora pela combinação linear das restrições fracas, dependendo do cumprimento das anteriores. Em cada combinação, seja forte ou fraca, são atribuídos pesos aplicados a cada restrição. O objetivo dos pesos é de indicar ao algoritmo que estes parâmetros requerem mais "protagonismo".

Importante notar uma diferença entre pesos e valor de custo. Os custos, incrementados a cada ocorrência de uma anormalidade, foram definidos em primero lugar, sendo que nunca foram alterados (indicados no capítulo 5 na secção de abordagem às restrições). Os pesos atuam sobre os custos já incrementados na sua totalidade, definidos após a verificação de testes de desempenho do cumprimento de restrições.

6.6.1 Restrições fortes

Apesar de todas as restrições fortes serem necessariamente cumpridas, existe uma hierarquia de relevância[2]. Numa primeira fase de estudo do desempenho do algoritmo, foram observadas de geração em geração quais as restrições que constituíam o gargalo, *bottleneck*, no seu cumprimento. As restrições identificadas foram:

- Exames em períodos adjacentes não podem ter estudantes em comum, devido à quantidade substancial de casos necessários a analisar (150);
- Garantir que, para qualquer calendário possível, o primeiro exame de época de recurso inicia-se posteriormente ao último exame de época normal, devido à dificuldade em "empurrar"os exames de recurso para uma data posterior aos exames de época normal, tendo em conta todos os calendários possíveis;
- Exames do mesmo ano não podem ocorrer no mesmo dia, devido à quantidade de *clusters* e a interdependência entre os mesmos;

É de senso comum que a restrição "Exames pertencentes ao mesmo período não podem ter estudantes em comum"é a mais importante, pois desafia as próprias leis da física. Dado isto, foi-lhe atribuída um custo elevado a cada ocorrência de modo a poder eliminar a ocorrência dessa anormalidade.

Posto isto, após a identificação das restrições foi-lhes atribuídas a cada um peso na escala de 1 a 10, sendo que a 1ª tem um peso multiplicativo de 10 e ambas a 2ª e 3ª um peso de 5. As restantes restrições têm associadas um peso unitário, 1.

6.6.2 Restrições fracas

Em relação às restrições fracas, visto o objetivo do problema ser maximizar o intervalo de tempo entre exames, o modo como os pesos foram definidos é idealizado diferentemente. Na perspetiva do estudante é preferível ter mais tempo para poder estudar durante a época normal para poder evitar a necessidade de ir ao exame de recurso.

Assim, apenas se atribuíram dois pesos não unitários, ambos na restrição "Intervalo de tempo entre a época normal e a época de recurso de cada *cluster* dentro dos limites". Esta restrição possui duas variáveis de custo, uma dedicada à época normal e outra à época de recurso, por isso serem atribuídos dois pesos à mesma restrição.

Com base na perspetiva do estudante, escolheu-se atribuir ao cenário de época normal um peso de 10 e para o cenário de época de recurso um peso de 5.

6.6.3 *Offset* na função objetivo

Visto que o melhor valor função objetivo obtido após a execução do algoritmo se resume à combinação linear das restrições, é importante poder observar à medida que se acompanha a

[2]Esta hierarquia não indica que uma restrição é mais importante que outra, mas sim na relevância prática de algoritmia do problema.

evolução do melhor *fitness*, a cada geração, em que momento é que o algoritmo conseguiu cumprir totalmente as restrições fortes e, por consequente, começou a contabilizar as restrições fracas na função objetivo.

Optou-se adicionar um desvio à combinação linear das restrições fracas, com o intuito de verificar graficamente a passagem descrita anteriormente, com um valor de -100 000. Este valor é, em primeiro lugar, negativo visto se tratar de um problema de minimização e não se pretende de todo prejudicar a função objetivo; em segundo lugar é elevado, e isto deve-se à probabilidade de, a partir do momento que as restrições fracas são analisadas, ocorrerem mais anormalidades do que casos expectáveis, sendo assim possível observar a função objetivo a decrescer continuamente, sendo o declive da reta que traça o valor da melhor função objetivo em função do número da geração ≤ 0.

A combinação linear relativa às restrições fortes, em linguagem de programação, está presente no anexo A, secção A.3, na linha 287. Quanto à combinação linear das restrições fracas, a linha correspondente no código é a 290.

6.7 Diagnóstico de simulação

Para poder analisar os resultados obtidos em cada instância, os dados necessários para análise estão inseridos em 2 ficheiros distintos, *stats* e *diagnóstico*, gerados ao final de cada execução. O primeiro inclui a evolução do melhor valor de função objetivo de cada geração; o segundo inclui as informações paramétricas e informações dos resultados obtidos, mais especificamente:

- Semente utilizada para gerar chaves de números aleatórios;
- O número de populações utilizadas;
- O tamanho de cada população;
- Número de *threads* utilizadas na descodificação;
- Fração da população a ser elite;
- Fração da população a ser mutante;
- Probabilidade de *offspring* (rho_e);
- Número de gerações em que as populações trocam de soluções elite entre si (*X_INTVL*);
- Número de soluções elite que as populações trocam entre si (*X_NUMBER*);
- Geração em que foi obtida a melhor solução;
- Geração em que as restrições fortes foram cumpridas;
- O melhor cromossoma em forma de vetor;

- Valor do melhor *fitness*;
- Os exames que ocorrem no mesmo dia que têm alunos em comum;
- Intervalos de tempo entre exame para cada cluster, para cada época de exames;

No anexo B, secção B.1, está disponível um exemplo deste ficheiro de diagnóstico.

O ficheiro *stats* é útil para poder instantaneamente numa folha de cálculo poder observar a curva de evolução da função objetivo (ferramenta introduzida na próxima secção). O ficheiro *diagnóstico* tem como objetivo fornecer ao utilizador informação básica acerca do resultado, podendo posteriormente com o auxílio de uma ferramenta fornecer mais dados como: qual a ordem dos exames nos *clusters* e qual o intervalo de tempo entre cada um dos exames; qual o calendário resultante no formato Data-Hora, com a possibilidade de exportar como ficheiro *CSV* para posterior importação no Google Calendar, por exemplo.

Além destes dois ficheiros, é gerado um terceiro, no formato folha de cálculo, que advém da funcionalidade extra do programa em alocar os exames nas salas, mencionado no capítulo 5 na secção 5.2.5. Um exemplo deste ficheiro está presente no anexo B, secção B.4.

Capítulo 7

Testes computacionais

Neste capítulo apresenta-se o desenho dos testes computacionais e uma ferramenta de análise para observar o resultado dos mesmos. É feita uma descrição dos diferentes testes e a finalidade que se espera alcançar em cada um dos testes. De seguida é introduzida a ferramenta que permite retirar conclusões dos dados recolhidos nos ficheiros-relatório de cada teste. São elaboradas análises estatísticas aos dados recolhidos em cada teste e conjuntos de testes. Por fim é realizada uma comparação entre a melhor solução obtida nos testes realizados com o mapa de exames construído manualmente no presente semestre do ano letivo 2016/2017. Todos os testes computacionais utilizaram os dados relativos ao presente semestre do ano letivo 2016/2017.

7.1 Plano de testes

Sendo este problema de caráter estocástico, significa que está submetido às "leis do acaso". Ao invés de problemas que possuem um único modo de evoluir, um problema estocástico tem uma indeterminação: mesmo que se conheçam as condições iniciais, existem, por vezes, infinitas direções nas quais o processo pode evoluir. O acaso está neste problema relacionado com a semente do gerador de números aleatório.

A avaliação do projeto é abordada em duas frentes: uma que servirá para observar o efeito da variação paramétrica dos parâmetros independentes na função objetivo, para uma dada semente do gerador de números aleatórios; outra que servirá para comparar os valores da função objetivo mantendo os parâmetros independentes fixos, variando o valor da semente.

A primeira frente será testada com uma amostra de 20 corridas, com sementes diferentes e a segunda será fixada o valor da semente, variando cada parâmetro na sua gama de variação apresentada na tabela 6.1(capítulo 6), mantendo os restantes parâmetros com os seus valores *standard*.

7.2 Ferramentas de análise

Para realizar a recolha de dados dos ficheiros e a organização dos mesmos para obter resultados diretos, implementou-se uma ferramenta, em Excel, que permite ao utilizador através de macros realizar a leitura de dados e o estudo dos mesmos sem qualquer esforço. Inclui também a funcionalidade para exportar o calendário resultante da solução em formato *CSV*, estruturado de modo a ser importado para o Google Calendar.

As macros utilizadas são:

- Leitura de ficheiro de diagnósticos: Lê os dados do ficheiro *diagnóstico*, cria o mapa de exames associado à solução onde analisa os intervalos de tempo entre exames pertencentes ao mesmo *cluster*, com a ordem das unidades curriculares, em cada época, em média.

Época Normal

	1		2		3		4		5			
		dias		dias		dias		dias				
Cluster1	AMAT2	7.00	FISI	2.33	CIRC		-		-			
Cluster2	ELEC1	4.67	TCON	6.00	SEEN		-		-			
Cluster3	EGES	4.67	SO	3.67	ELEC3	3.67	PDSI		-			
Cluster4	CDIG	2.33	EGES	5.00	EIND	2.33	IELE	4.33	SINF			
Cluster5	EGES	4.33	RTDI	3.67	MELE	3.67	SINF	2.33	RESEE			
Cluster6	COPT	3.67	APRO	3.67	IOPE	3.67	STEL		-			
Cluster7	PGRE	1.33	COMO	5.00	IOPE	3.67	STEL		-			
Cluster8	SMUL	4.00	IOPE	3.67	STEL	5.00	PCIM		-			
Cluster9	PCVL	4.00	IOPE	3.67	STEL		-		-			
Cluster10	SADE	4.00	SQFI	8.00	SBIC		-		-			
Cluster11	PCVL	12.00	SBIC		-		-		-			
Cluster12	ACCM	4.00	APS	5.00	SBIC		-		-			
Cluster13	RIND	8.00	APS	5.00	SBIC		-		-			
Cluster14	PLIC	3.67	DOIC	4.67	MQUA		-		-			
Média		4.83333		4.5641		3.6667		3.3333			Média overall	4.09941

Figura 7.1: Exemplo de *clusters* ordenados e tempo médio entre cada exame, na ferramenta Excel

- Leitura de *stats*: Lê os dados do ficheiro *stats*, onde recolhe os valores da função objetivo e gera dois gráficos onde é possível observar a evolução do *fitness* ao longo das gerações e, noutra perspetiva, numa escala logarítmica é possível observar a evolução do *fitness* até ao momento em que se cumprem as restrições fortes do problema.

(a) Evolução do *fitness* ao longo da execução do algoritmo, na ferramenta Excel

(b) Evolução do *fitness* no cumprimento das restrições fortes, na ferramenta Excel

Figura 7.2: Resultados obtidos com arrefecimento simulado

- Guardar ficheiro: Guarda a folha de cálculo preenchida com um nome personalizado pelo utilizador para futuras observações.
- Exportar calendário: Permite exportar a folha de cálculo associada ao calendário, no formato Data-Hora, para um ficheiro *CSV*.

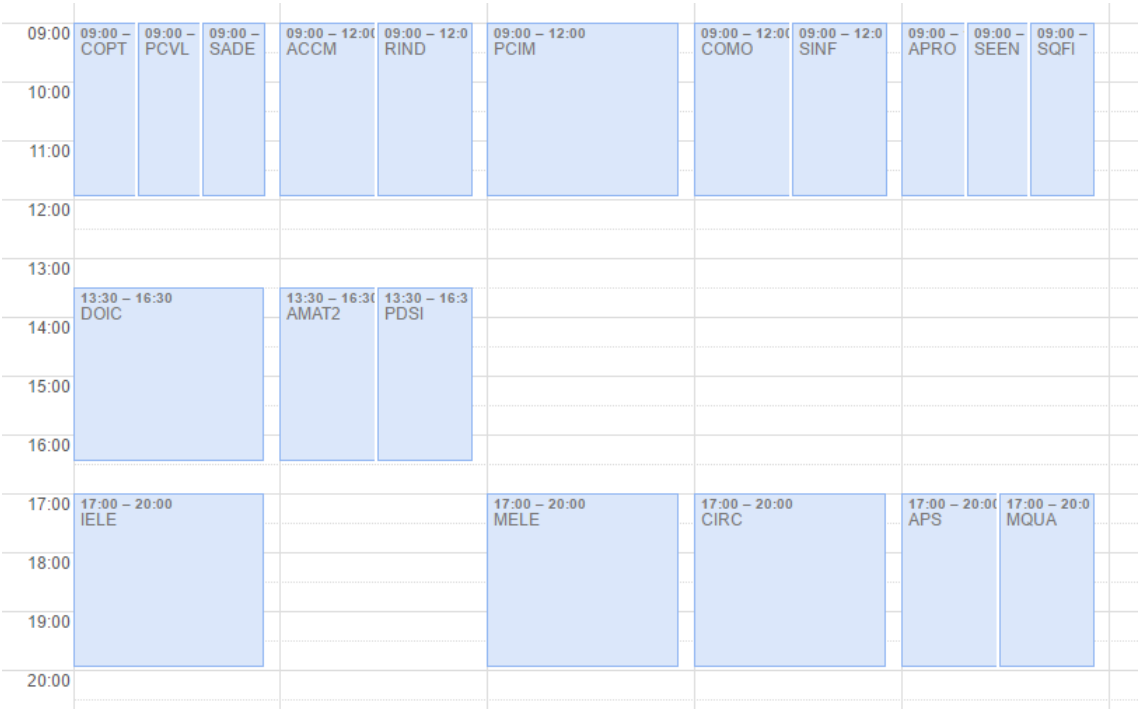


Figura 7.3: Exemplo de mapa importado para Google Calendar

7.3 Análise Estatística

Na primeira abordagem à validação do modelo, a que corresponde a execução de várias corridas com valores de semente diferentes em cada uma, são recolhidos os dados: geração em que restrições fortes são cumpridas; geração em que é obtida a melhor solução e o seu valor de *fitness*. Obteve-se a seguinte tabela 7.1.

Tabela 7.1: Dados das amostras com variação do valor da semente. Serve a legenda: GRFC- Geração em que Restrições Fortes são Cumpridas, GMS- Geração em que Melhor Solução é obtida.

Amostra	Semente	GRFC	GMS	Fitness
1	992	511	2600	-123886
2	1905	345	2966	-109865
3	2058	301	2972	-126087
4	2244	906	2825	-129873
5	2356	728	2600	-114881
6	2959	443	1859	-116863
7	3452	465	2347	-113893
8	3836	266	1962	-114074
9	3952	523	2972	-111095
10	4453	313	2987	-107890
11	4545	272	1655	-103091
12	4777	308	2303	-120096
13	5711	328	1924	-114892
14	7039	388	2782	-129893
15	7915	418	2646	-110897
16	7930	305	2305	-107880
17	8707	294	1920	-119895
18	8847	312	2459	-134885
19	8947	376	1793	-113886
20	8950	475	1802	-119880

A partir destes dados foram construídos dados estatísticos relativamente à média, desvio padrão, máximo e mínimo, presentes na tabela 7.2.

Tabela 7.2: Resultados estatísticos da primeira abordagem, para uma amostra N=20.

Objeto de análise	μ	σ	Máximo	Mínimo
Geração em que restrições fortes são cumpridas	414	162	906	266
Geração em que a melhor solução é atingida	2384	460	2987	1655
Fitness (Restrições Fortes)	0	0	0	0
Fitness (Restrições Fracas)	-117185.10	8356.84	-103091	-134885

Em cada semente foi também observada qual a média de intervalo de tempo entre exame, em cada *cluster*, em cada época de recurso, que resultou na tabela seguinte.

Tabela 7.3: Intervalos de tempo médio entre exames do mesmo *cluster*, para diferentes sementes. Serve a legenda: ITM - Intervalo de Tempo Médio

Sementes	ITM- Normal	ITM- Recurso
992	3.71	3.34
1905	3.32	4.29
2058	3.89	2.93
2244	3.85	2.95
2356	3.98	3.44
2959	3.88	3.13
3452	3.70	3.35
3836	3.27	3.59
3952	3.58	3.65
4453	3.72	3.64
4545	2.76	3.26
4777	4.17	3.21
5711	4.14	3.48
7039	4.17	3.60
7915	4.50	3.66
7930	4.26	4.16
8707	3.95	3.26
8847	3.63	3.09
8947	3.58	3.41
8950	3.93	3.65

Como esperado, os dados obtidos comprovam o constatado em 7.1, cada corrida com a sua própria semente obteve funções objetivo únicas, sem alterar os valores dos parâmetros, observável na tabela 7.1.

Como se pode constatar pela tabela 7.2, em todas as corridas as restrições fortes são cumpridas na sua totalidade, em média a partir da geração 414. Quanto ao valor da função objetivo, tanto a melhor solução (-134885), como a pior solução (-103091), o que indica, ao ser inferior ao valor de desvio aplicado na função (recorde-se -100000), que há maior ocorrência de exames bem espaçados que exames mal espaçados entre si.

Complementando a tabela 7.2 com a tabela 7.3, pode-se concluir que a melhor função objetivo, não indica o mapa com o melhor espaçamento entre exames, o que é subentendido por esta se tratar de uma combinação linear de diversas variáveis. A melhor função objetivo, segundo a média de intervalo de tempo entre exames de época normal (maior protagonista da combinação linear), coloca-se no 17º de entre as 20 amostras.

Relativamente à segunda abordagem de estudo de desempenho do modelo, foram realizados testes onde é feita uma variação paramétrica individual, em que os restantes parâmetros mantêm os seus valores, com o objetivo de verificar de que modo afeta a evolução do algoritmo.

Para facilitar a análise dos dados recolhidos e poder comparar resultados, colocaram-se todos os dados numa única tabela. A tabela 7.4 inclui os dados relativos ao parâmetro variado, o valor que toma o parâmetro, o intervalo de tempo entre exames (em média) na época normal e de recurso, a geração a que se cumprem as restrições fortes, a geração onde se obtém a melhor solução da corrida, o valor de *fitness* associado e o número de casos em que há estudantes com mais que um exame por dia (a unidades curriculares com pelo menos 2 anos curriculares de diferença).

Tabela 7.4: Dados dos testes realizados com variação paramétrica, usando a mesma semente. Serve a legenda: ITM- Intervalo de Tempo Médio, GRFC- Geração em que Restrições Fortes são Cumpridas, GMS- Geração em que Melhor Solução é obtida.

Parâmetro	Valor	ITM-Normal	ITM-Recurso	GRFC	GMS	Fitness	Casos
Rho_e	60%	3.77	3.75	332	2062	-110883	2
Rho_e	70%	3.89	2.93	301	2972	-126087	5
Rho_e	80%	4.12	3.07	532	2439	-118891	3
Rho_e	90%	3.68	3.52	380	1850	-122874	5
p_e	10%	3.89	3.88	249	2597	-128090	4
p_e	15%	3.86	2.92	202	2945	-112892	4
p_e	20%	3.89	2.93	301	2972	-126087	5
p_e	25%	3.60	2.95	383	2614	-115892	2
p_e	30%	3.89	3.71	289	2802	-127884	5
p_m	10%	3.06	3.48	844	2952	-107881	7
p_m	15%	3.89	2.93	301	2972	-126087	5
p_m	20%	3.74	3.93	390	1761	-125894	3
p_m	25%	3.63	4.17	579	2944	-123895	3
p_m	30%	3.48	3.24	825	2746	-131874	6
X_NUMBER	1	3.68	3.51	560	2718	-119889	3
X_NUMBER	2	3.89	2.93	301	2972	-126087	5
X_NUMBER	3	3.83	3.30	303	2693	-109868	6
X_INTVL	50	4.19	3.53	521	1727	-118882	5
X_INTVL	60	4.39	3.11	403	2846	-124882	4
X_INTVL	70	4.41	3.60	538	2545	-107854	4
X_INTVL	80	4.41	3.11	298	2695	-107090	4
X_INTVL	90	3.41	3.22	521	1838	-103876	8
X_INTVL	100	3.89	2.93	301	2972	-126087	5

De acordo com a tabela 7.4 pode-se extrair a informação estatística relativa à média, ao desvio padrão e os valores máximos e mínimos.

Através da análise da tabela 7.5, confirma-se a validação quanto ao cumprimento das restrições fortes para toda a variação paramétrica, o que era de esperar, em média a partir da geração 420. Do mesmo modo que na análise anterior dos dados obtidos através de diferentes sementes, todos as soluções indicam que há maior ocorrência de exames bem espaçados do que exames mal

Tabela 7.5: Resultados estatísticos de média, desvio padrão, máximo e mínimo da segunda abordagem.

Objeto de análise	μ	σ	Máximo	Mínimo
Geração em que restrições fortes são cumpridas	420	171	844	202
Geração em que a melhor solução é atingida	2593	434	2972	1727
Fitness(Restrições Fortes)	0	0	0	0
Fitness(Restrições Fracas)	-119553.30	8279.18	-103876	-131874
Casos	4.48	1.47	8	2

espaçados. Verifica-se também que o intervalo de tempo médio entre exames não é proporcional ao decrescimento do valor de *fitness*, justificado na análise anterior.

Esta abordagem permite ao utilizador qual o impacto que os parâmetros têm no problema, permitindo assim compreender que valores são mais apropriados aos objetivos do problema. Num exemplo prático, se o utilizador pretender obter o melhor tempo médio entre exames de época normal, os valores de cada parâmetro mais apropriados seriam:

- Rho_e - 70%;
- p_m - 15%;
- X_INTVL- 70;
- p_e - 10%;
- X_NUMBER - 2;

Após a execução do algoritmo, com a mesma semente dos testes anteriores e com os parâmetros acima definidos, obtiveram-se os seguintes resultados:

Tabela 7.6: Resultados do teste com parâmetros redefinidos. Serve a legenda: ITM- Intervalo de Tempo Médio, GRFC- Geração em que Restrições Fortes são Cumpridas, GMS- Geração em que Melhor Solução é obtida, RFo - Restrições Fortes, RFr - Restrições Fortes.

GRFC	GMS	ITM- Normal	ITM- Recurso	Fitness (RFo)	Fitness(RFr)	Casos
272	2840	3.39	3.51	0	-117891	5

No caso em que os parâmetros são independentes entre si, a seleção dos melhores valores com base num objetivo implica um melhoramento no processo. Como se demonstra neste exemplo prático, não surgiu nenhuma melhoria no intervalo de tempo médio entre exames de época normal. Isto leva a concluir que os parâmetros não são independentes entre si.

Em trabalhos futuros, para obter um estudo adequado sobre quais os valores dos parâmetros que se deve utilizar para melhorar um processo é necessário recorrer a um DOE, *Design of Experiments*. Este método permite verificar a relação entre os diferentes parâmetros e verificar como é que os mesmos variam a função objetivo.

7.4 Comparação com calendário atual

Mesmo cumprindo todos os objetivos a que o modelo se propõe, é necessário fazer uma última análise comparando com o calendário atual do presente semestre do ano letivo elaborado manualmente pelo DEEC.

Esta análise será descrita pelo cumprimento das restrições[1], o número de casos em que há estudantes com mais que um exame por dia, o intervalo médio entre exames em cada época e, por fim, o valor de *fitness*.

Tabela 7.7: Resultados da análise do calendário atual, serve a legenda: NC - Não Cumpridas, ITM- Intervalo de Tempo Médio.

Objeto de análise	Restrições NC	Casos	ITM - Normal	ITM - Recurso	Fitness
Calendário manual	1	12	4.97	3.58	12997
Melhor solução	0	6	3.63	3.09	-134885

Comparando o calendário elaborado manualmente e a melhor solução obtida pelo modelo

Como se pode verificar pela tabela 7.7, no calendário manual uma das restrições fortes não foi cumprida, que corresponde a "Exames pertencentes ao mesmo período não podem ter estudantes em comum", já na melhor solução obtida o mesmo não acontece. Os exames em causa são Eletrónica 1 e Comunicações Móveis, que têm um estudante em comum. A ocorrência de doze casos em que estudantes têm mais que um exame por dia é substancialmente superior a todos os testes realizados com o modelo implementado (máximo de 8), e no caso da melhor solução 6. O intervalo médio de tempo entre exames na época normal é superior aos obtidos nos testes computacionais num valor de 0,49. O valor de *fitness* superior a 0 indica que o calendário resultante é impraticável segundo as restrições do problema.

Esta análise permite validar a robustez do modelo implementado quanto ao cumprimento de todas as restrições fortes do problema.

[1] As restrições do modelo foram estabelecidas segundo as diretrizes da equipa que elaborou o calendário manual, podendo o entanto não ter sido integralmente cumpridas pela própria equipa.

Capítulo 8

Conclusões e Trabalho Futuro

Neste capítulo discute-se o cumprimento dos objetivos inicialmente propostos, sugerindo ainda ideias para trabalhos futuros acerca deste tema.

8.1 Conclusões e Trabalho Futuro

Nesta dissertação foi apresentada uma abordagem à produção de um mapa de exames no ensino superior utilizando como ferramenta um algoritmo genético de chaves aleatórias viciadas.

A conceção da abordagem de resolução resultou de uma confluência entre o que é geralmente abordado na literatura e as diretrizes do curso de Engenharia Electrotécnica e de Computadores, comprometendo-se a cumprir um conjunto de objetivos. A revisão bibliográfica revelou-se uma fase importante deste trabalho, permitindo facilitar a compreensão do problema e consolidar as perspetivas de vários investigadores, neste tema de calendarização de exames.

Após realização de testes computacionais com dados reais, pode-se concluir que os resultados foram mais que satisfatórios. Os testes computacionais demoraram em média cerca de uma hora e quarenta minutos cada.

O desempenho do algoritmo implementado foi avaliado considerando variação paramétrica e através da comparação com o calendário elaborado manualmente pelo corpo docente responsável, verificando-se a sua robustez.

Este trabalho será de grande ajuda para poder gerar um calendário que cumpra as restrições consideradas, em menos de duas horas, servindo como ponto de partida e poupando tempo e esforço. Um exemplo de calendário de exames gerado por este trabalho está no anexo C.

Como sugestões para trabalhos futuros relacionados com este tema, sugeria as seguintes pontos:

- Consideração de feriados;
- Introdução de conceito de complexidade de unidade curricular e influência no mapa de exames;

- Uso de taxa de reprovação de cada unidade curricular para definir a importância de cada exame;
- Uso de heurísticas de reconstrução nos cromossomas;
- Introduzir conceito de reinicialização do algoritmo com base no número de iterações/gerações à solução "alvo".

Como nota final, é imprescindível de referir que foi importante para o autor o trabalho intelectual que esta dissertação exigiu. As capacidades de compreensão e perceção de problema na área da Investigação Operacional foram bastante enriquecidas, especialmente na campo de calendarização e sequenciamento.

Anexo A

Código *framework*

A.1 Função Main

```
1
2
3  #include <iostream>
4  #include <vector>
5  #include <fstream>
6  #include <iomanip>
7  #include "SampleDecoder.h"
8  #include "MTRand.h"
9  #include "BRKGA.h"
10 #include "instancedata.h"
11 #include <chrono>
12
13
14
15 bool pairCompare(const std::pair<vector<double>, double>& firstElem, const std::pair<
       vector<double>, double>& secondElem) {
16  return firstElem.second > secondElem.second;
17
18 }
19
20 bool pairCompare4(const std::pair<int, int>& firstElem, const std::pair<int,int>&
       secondElem) {
21  return firstElem.second > secondElem.second;
22
23 }
24
25  int weekend_account( int period){
26  int retorno = period;
27  int account = 0;
28  while (1) {
29   retorno -= 15;
30   if (retorno > 0)
31    account++;
32   else break;
33  }
34  return period + account * 6;
35 }
```

```
36
37 bool pairCompare2(const std::pair<int, int>& firstElem, const std::pair<int, int>&
       secondElem) {
38  return firstElem.second < secondElem.second;
39
40 }
41
42 bool pairCompare3(const std::pair<string, int>& firstElem, const std::pair<string, int>&
       secondElem) {
43  return firstElem.second > secondElem.second;
44
45 }
46
47 int main(int argc, char* argv[]) {
48  const unsigned n = 2*36;  // size of chromosomes
49  const unsigned p = 100; // 100 // size of population
50  const double pe = 0.20; //0.20 // fraction of population to be the elite-set
51  const double pm = 0.15; //0.15 // fraction of population to be replaced by mutants
52  const double rhoe = 0.70; // probability that offspring inherit an allele from elite
       parent
53  const unsigned K = 2;  // number of independent populations
54  const unsigned MAXT = 2; // number of threads for parallel decoding
55  const unsigned sem = 5;     // numero de semanas para marcar exames
56  char filename[50] = "src/matriz2.txt";
57  using namespace std::chrono;
58  high_resolution_clock::time_point t1 = high_resolution_clock::now();
59
60  InstanceData Data(filename);
61
62
63  SampleDecoder decoder(Data);   // initialize the decoder
64
65  const long unsigned rngSeed = 2058; // seed to the random number generator
66  MTRand rng(rngSeed);     // initialize the random number generator
67
68  // initialize the BRKGA-based heuristic
69  BRKGA< SampleDecoder, MTRand > algorithm(n, p, pe, pm, rhoe, decoder, rng, K, MAXT);
70
71  unsigned generation = 0;  // current generation
72  const unsigned X_INTVL = 100; // exchange best individuals at every 100 generations
73  const unsigned X_NUMBER = 2; // exchange top 2 best
74  const unsigned MAX_GENS = 3000; // run for 1000 gens
75
76  vector<pair<vector<double>, double>> lista2;
77  fstream statistics;
78  statistics.open("src/stats.txt", fstream::app);
79
80  if (statistics.fail()) {
81   cout << "something wrong with input file: " << endl;
82   cout << "pgm will terminate now " << endl;
83   exit(0);
84  }
85
86
87
88  int r = 0;
89  int critical_gen;
```

```
90  double follow_best=99999999;
91  int strong_satisfied;
92  int strong_flag = 0;
93  do {
94   algorithm.evolve(); // evolve the population for one generation
95
96   if((++generation) % X_INTVL == 0) {
97    algorithm.exchangeElite(X_NUMBER); // exchange top individuals
98   }
99   cout<<generation << " \t " << algorithm.getBestFitness() << endl;
100
101   statistics << (double) algorithm.getBestFitness() << endl;
102
103
104   if (algorithm.getBestFitness() <= 0 && strong_flag == 0) {
105    strong_satisfied = generation;
106    strong_flag = 1;
107   }
108
109   if (algorithm.getBestFitness() < follow_best) {
110    follow_best = algorithm.getBestFitness();
111    critical_gen = generation;
112   }
113
114
115   //reset situation
116   if (algorithm.getBestFitness() > 0 && generation == MAX_GENS-1) {
117    generation = 0;
118   }
119
120
121
122
123  } while (generation < MAX_GENS);
124  statistics << "----------------------------------------------------------" << endl;
125  statistics.close();
126
127  std::cout << "Best solution found has objective value = "
128     << algorithm.getBestFitness() << std::endl;
129
130  std::cout << " The best solution is : \n";
131
132   // imprimir a solucao final
133
134  vector<double> solution = algorithm.getBestChromosome();
135  int i = 1;
136  for (vector<double>::iterator it = solution.begin(); it != solution.end();it++, i++)
137   cout << i << "-> " << ceil(*it * sem * 5 * 3) << endl;
138
139
140  fstream file;
141
142  file.open("src/diagnostico.txt", fstream::app);
143  time_t timestamp;
144
145  if (file.fail()) {
146   cout << "something wrong with input file: " << endl;
```

```
147     cout << "pgm will terminate now " << endl;
148     exit(0);
149    }
150
151
152    file << "Diagnostico efetuado em " << time(&timestamp) << endl;
153    file << "Semente utilizada: " << rngSeed << endl;
154    file << "Populacoes utilizadas: " << K << endl;
155    file << "Tamanho da populacao: " << p << endl;
156    file << "Threads utilizadas: " << MAXT << endl;
157    file << "Fracao da populacao a ser elite: " << pe << endl;
158    file << "Fracao mutantes: " << pm << endl;
159    file << "Probabilidade offspring: " << rhoe << endl;
160    file << "Troca de cromossomas elite a todas as : " << X_INTVL << " geracoes" << endl;
161    file << "Numero de cromossomas elite a trocar: " << X_NUMBER << endl;
162
163    file << "A solucao otima foi obtida em: " << critical_gen << endl;
164
165    file << "As restricoes fortes foram cumpridas a partir de: " << strong_satisfied <<
           endl;
166
167    file << "\n \n" << "Melhor solucao: " << endl;
168
169    for (vector<double>::iterator it = solution.begin(); it != solution.end();it++)
170     file << ceil((*it)*sem*5*3) << endl;
171
172
173    file << "\n" << "Com fitness de: " << algorithm.getBestFitness() << endl;
174
175
176
177    //////////////////////////////////////Avaliacao da solucao
           ////////////////////////////////////////////////////////////////
178
179    std::vector<vector<int>> time(sem * 5 * 3);
180
181
182    int start_day = 29;
183    int month_last_day = 31;
184    string month_start = "May";
185    string month_next = "June";
186
187
188
189    //cout << time.size();
190    float escala = (float)time.size();
191
192    int pos;
193    // criar vetor de tempos
194    for (int i = 0; i < algorithm.getBestChromosome().size();i++) {
195     pos = ceil(algorithm.getBestChromosome().at(i)*escala); // ceil para fazer round ao
           proximo inteiro evita problemas para o periodo 1 e periodo 60
196     time.at(pos - 1).push_back((int)(i / 2));
197     i++;
198     pos = ceil(algorithm.getBestChromosome().at(i)*escala); // ceil para fazer round ao
           proximo inteiro evita problemas para o periodo 1 e periodo 60
199     time.at(pos - 1).push_back((int)(i / 2));
```

```
200 }
201
202 // LEITURA DO VETOR DE TEMPOS
203
204 /*for (int i = 0; i < time.size();i++) {
205 cout << i+1 << " - ";
206 for (int j = 0; j < time.at(i).size();j++) {
207 cout << " " << time.at(i).at(j); //o +1 e' porque os valores estao com as posicoes dos
        vetores, nao comecam em 1 mas em 0
208 }
209 cout << endl;
210 }*/
211
212 // Restricao 0 - Nao pode haver mais que 3 exames por periodo
213
214
215
216 for (int k = 0; k < time.size();k++) {
217  if (time.at(k).size() > 3) {
218   string month;
219   int print = start_day + ceil(k / 3) - 1;
220   if (print > 31 && print <= 61) {
221    print = print - 31;
222    month = "June";
223   }
224   if (print > 31 && print > 61) {
225    print = print - 61;
226    month = "July";
227   }
228   if (!(month.empty())) {
229    cout << "\nNo dia " << print << " de " << month << "existem " << time.at(k).size()
         << "exames.\n";
230    file << "\nNo dia " << print << " de " << month << "existem " << time.at(k).size()
         << "exames.\n";
231
232   }
233
234   else {
235    cout << "\nNo dia " << print << " de " << month_start << "existem " << time.at(k).
         size() << "exames.\n";
236    file << "\nNo dia " << print << " de " << month_start << "existem " << time.at(k).
         size() << "exames.\n";
237   }
238  }
239
240 }
241
242
243
244 /*
     ********************************************************************************************
     */
245
246 // 1 Restricao - Exames que ocorrem em cada periodo nao podem ter estudantes em comum
247
248 // inicializar o custo a zero
249 size_t size; // valor que da o tamanho de cada vetor na matriz de tempo
```

```
250  for (int k = 0; k < time.size();k++) // percorrer todos os vetores da matriz
251  {
252   size = time.at(k).size();
253   int m = 0;
254   if (size > 1) {
255    for (std::vector<int>::iterator it = time.at(k).begin(); m< time.at(k).size() - 1;m
         ++, it++) // todas as combinacoes entre as posicoes do vetor
256    {
257
258     std::vector<int>::iterator it_next; // apontador ao objeto seguinte
259     it_next = it;
260     it_next++; // comeca sempre uma posicao a frente do apontador principal
261     int obj = *it;
262     for (; it_next != time.at(k).end(); it_next++)
263     {
264      int obj2 = *it_next;
265      if (obj != obj2) {
266       if (Data.getconflito(obj, obj2) != 0) {
267        string obj_name = Data.getnameofexam(obj);
268        string obj2_name = Data.getnameofexam(obj2);
269        cout << obj_name << " e " << obj2_name << " ocorrem ao mesmo tempo.\n";
270        file << obj_name << " e " << obj2_name << " ocorrem ao mesmo tempo.\n";
271
272       }
273
274
275      }
276
277
278    }
279   }
280
281  }
282
283 }
284
285 // 2a Restricao Garantir que nenhum estudante tem exames em periodos adjacentes
286
287
288
289 int gap = 1;
290 int custo2 = 0;
291 int threshold = 0;
292 for (int g = 0, count = 0; g < time.size() - 2; g++) { //percorrer o vetor de tempos
293  vector<int> period_vector2;
294  vector<int> period_vector3;
295  vector<int> period_vector4;
296  for (int t = 0; t < time.at(g).size();t++) {
297   period_vector2.push_back(time.at(g).at(t));
298  }
299  for (int y = 0; y < time.at(g + 1).size();y++) {
300   period_vector3.push_back(time.at(g + 1).at(y));
301  }
302  for (int z = 0; z < time.at(g + 2).size();z++) {
303   period_vector4.push_back(time.at(g + 2).at(z));
304  }
```

```
305   if (period_vector4.empty() != 0 && (period_vector2.empty() != 0 || period_vector3.
          empty() != 0)) {
306    for (std::vector<int>::iterator ti = period_vector3.begin(); ti != period_vector3.end
          (); ti++) {
307     std::vector<int>::iterator ti4 = period_vector4.begin();
308     std::vector<int>::iterator ti2 = period_vector2.begin();
309     for (; ti4 != period_vector4.end();ti4++) {
310      if (*ti != *ti4) {
311       if (Data.getconflito(*ti, *ti4) != 0) {
312        string obj_name = Data.getnameofexam(*ti);
313        string obj2_name = Data.getnameofexam(*ti4);
314        cout << obj_name << " e " << obj2_name << "  tem os seus exames em periodos
              consecutivos.\n";
315        file << obj_name << " e " << obj2_name << "  tem os seus exames em periodos
              consecutivos.\n";
316       }
317      }
318      else {
319       string obj_name = Data.getnameofexam(*ti);
320       cout << obj_name << " tem os seus exames em periodos consecutivos.\n";
321       file << obj_name << " tem os seus exames em periodos consecutivos.\n";
322
323      }
324     }
325     for (; ti2 != period_vector2.end();ti2++) {
326      if (*ti != *ti2) {
327       if (Data.getconflito(*ti, *ti2) != 0) {
328        string obj_name = Data.getnameofexam(*ti);
329        string obj2_name = Data.getnameofexam(*ti2);
330        cout << obj_name << " e " << obj2_name << "  tem os seus exames em periodos
              consecutivos.\n";
331        file << obj_name << " e " << obj2_name << "  tem os seus exames em periodos
              consecutivos.\n";
332       }
333      }
334      else {
335       string obj_name = Data.getnameofexam(*ti);
336       cout << obj_name << " tem os seus exames em periodos consecutivos.\n";
337       file << obj_name << " tem os seus exames em periodos consecutivos.\n";
338      }
339     }
340    }
341   }
342
343
344   if (period_vector2.empty() == 0 && period_vector4.empty() == 0) {
345    std::vector<int>::iterator ti4 = period_vector4.begin();
346    std::vector<int>::iterator ti2 = period_vector2.begin();
347    for (;ti2 != period_vector2.end();ti2++) {
348     for (;ti4 != period_vector4.end();ti4++) {
349      if (*ti2 != *ti4) {
350       if (Data.getconflito(*ti2, *ti4)>threshold)
351       {
352        string obj_name = Data.getnameofexam(*ti4);
353        string obj2_name = Data.getnameofexam(*ti2);
354        cout << obj_name << " e " << obj2_name << "  tem os seus exames no mesmo dia.\n";
355        file << obj_name << " e " << obj2_name << "  tem os seus exames no mesmo dia.\n";
```

```
356         }
357        }
358        else {
359         string obj_name = Data.getnameofexam(*ti2);
360         cout << obj_name << " tem os seus exames no mesmo dia.\n";
361         file << obj_name << " tem os seus exames no mesmo dia.\n";
362
363        }
364       }
365      }
366     }
367     g++;
368     g++;
369    }
370
371    // 3 restricao - avaliacao do espacamento temporal entre clusters
372    int rectify;
373    size_t size_cluster = Data.getsizeofclustermatrix();
374    //epoca normal
375    cout << "\nNORMAIS\n\n";
376
377    for (int i = 0; i < size_cluster;i++) {
378     vector<int> cluster = Data.getcluster(i);
379     vector<int> pos;
380     cout << "\n\nEspacamento temporal nos clusters durante epoca normal:\n\n";
381     file << "\n\nEspacamento temporal nos clusters durante epoca normal:\n\n";
382     for (int j = 0; j < cluster.size();j++) {
383      rectify = weekend_account(ceil(algorithm.getBestChromosome().at(cluster.at(j) * 2)*
           escala));
384      pos.push_back(rectify);
385       // x2 por cada par de genes corresponde a um unico exame
386     }
387     sort(pos.begin(), pos.end());
388     cout << "Cluster " << i << "- Os espacamentos sao: ";
389     file << "Cluster " << i << "- Os espacamentos sao: ";
390     for (int k = 0; k < cluster.size() - 1;k++) {
391      cout <<(float) (pos.at(k + 1) - pos.at(k))/3 << " ";
392      file << (float)(pos.at(k + 1) - pos.at(k)) / 3 << " ";
393     }
394     cout << "dias.\n";
395     file << "dias.\n";
396
397    }
398
399    cout << "\nRECURSOS\n\n";
400    file << "\nRECURSOS\n\n";
401    //epoca recurso
402    for (int i = 0; i < size_cluster;i++) {
403     vector<int> cluster = Data.getcluster(i);
404     vector<int> pos;
405     cout << "\n\n Espacamento temporal nos clusters durante epoca recurso:\n\n";
406     file << "\n\n Espacamento temporal nos clusters durante epoca recurso:\n\n";
407
408     for (int j = 0; j < cluster.size();j++) {
409      rectify = weekend_account(ceil(algorithm.getBestChromosome().at(cluster.at(j) * 2+1)*
           escala)); // +1 para aceder ao gene correspondente ao recurso
410      pos.push_back(rectify);
```

```
411
412     }
413     sort(pos.begin(), pos.end());
414     cout << "Cluster " << i << "- Os espacamentos sao: ";
415     file << "Cluster " << i << "- Os espacamentos sao: ";
416
417     for (int k = 0; k < cluster.size() - 1;k++) {
418
419      cout << (float) (pos.at(k + 1) - pos.at(k))/3 << " ";
420      file << (float)(pos.at(k + 1) - pos.at(k)) / 3 << " ";
421
422     }
423     cout << "dias.\n";
424     file << "dias.\n";
425   }
426
427
428   cout << "A solucao otima foi obtida em: " << critical_gen << endl;
429   cout << "As restricoes fortes foram cumpridas a partir de: " << strong_satisfied <<
          endl;
430
431   cout << "-------------------------\nFimda avaliacao\n-------------------------\nPressione
          enter para encerrar o terminal.\n\n\n";
432   file << "||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||" << endl;
433
434   file.close();
435 }
```

A.2 Leitor de dados

A.2.1 Funções

```
1
2  #ifndef _INSTANCE_DATA_H_
3  #define _INSTANCE_DATA_H_
4
5  #pragma once
6
7  #include<iostream>
8  #include<fstream>
9  #include<cstdlib>
10 #include<cstring>
11 #include<vector>
12 #include<cmath>
13 #include<string>
14
15
16 using namespace std;
17
18 class InstanceData
19 {
20  //friend class SampleDecoder;
21
22 private:
23  vector<vector<int>> matrix;
24  vector<vector<int>> cluster;
25  vector<string> name;
26  vector<vector<int>> cluster2;
27
28 public:
29  //construtor
30  InstanceData();
31  InstanceData(const char * dados);
32  //destruidor
33  ~InstanceData();
34
35  int getconflito(int x, int y) const { return matrix.at(x).at(y); }; // vai buscar o
        valor da matriz na posicao x e y
36  size_t getsizeofmatrix() const { return matrix.size(); };
37
38  vector<int> getlinhadeconflito(int x) const { return matrix.at(x); };
39
40  vector<int> getcluster(int k) const { return cluster.at(k); };
41
42  size_t getsizeofclustermatrix() const { return cluster.size(); };
43
44  int samecluster(int e1, int e2) const {
45   int flag1 = 0;
46   int flag2 = 0;
47   size_t cluster_size = getsizeofclustermatrix();
48   for (int i = 0; i < cluster_size;i++) {
49    vector<int> cluster = getcluster(i);
50    for (int j = 0; j < cluster.size();j++) {
51     if (e1 == cluster.at(j))
```

```
52        flag1 = 1;
53       if (e2 == cluster.at(j))
54        flag2 = 1;
55      }
56      if (flag1 == 1 && flag2 == 1)
57       return 1;
58      flag1 = 0;
59      flag2 = 0;
60
61    }
62    return 0;
63
64   };
65
66
67   string getnameofexam(int id) const { return name.at(id); };
68
69   size_t getsizeofcluster2() const { return cluster2.size(); };
70
71   size_t getsizeofcluster2_element(int x) const { return cluster2.at(x).size(); };
72
73   int getvalueofcluster2(int row, int col) const { return cluster2.at(row).at(col); };
74
75   void getmatch(int e, int* c1, int* c2, int* c3)const  {
76    size_t sizeofcluster2 = getsizeofcluster2();
77    int found = 0;
78    for (int i = 0; i < sizeofcluster2;i++) {
79     size_t sizeofcluster2_x = getsizeofcluster2_element(i);
80     for (int j = 0; j < sizeofcluster2_x;j++) {
81      int aux;
82      aux = getvalueofcluster2(i, j);
83      if (e == aux) {
84       found = 1;
85       if (i == 0)
86        (*c1)++;
87       if (i == 1)
88        (*c2)++;
89       if (i == 2)
90        (*c3)++;
91       return;
92      }
93     }
94    }
95   };
96  };
97
98  #endif
```

A.2.2 Código

```
1
2  #include "instancedata.h"
3  #include <fstream>
4
5  InstanceData::InstanceData()
6  {
7  }
8
9  //construtor geral
10 InstanceData::InstanceData(const char * dados)
11 {
12
13  ifstream myReadFile;
14
15  myReadFile.open(dados);
16
17  if (myReadFile.fail()) {
18   cout << "something wrong with input file: " << dados << endl;
19   cout << "pgm will terminate now " << endl;
20   exit(0);
21  }
22
23  int coluna=0; // Dimensao da matriz, coluna x coluna
24  int clusterx = 0; // numero de clusters
25  int clustery = 0; // dimensao de cada clusters
26  myReadFile >> coluna;
27  //cout << coluna << endl;
28
29
30  matrix.resize(coluna);
31
32  int number;
33
34  //percorre o ficheiro
35  for (int i = 0; i < coluna; i++) {
36   for (int j = 0; j < coluna; j++) {
37    myReadFile >> number;
38    matrix.at(i).push_back(number);
39   }
40
41  }
42  cout << "\n \n \n \n \n A CARREGAR DADOS \n \n \n \n \n" << endl;
43
44
45  for (int i = 0; i < coluna; i++) {
46
47   cout << " Linha " << i;
48   for (int j = 0; j < coluna; j++) {
49    cout << " " << matrix.at(i).at(j) << " ";
50    // PARA OBSERVAR
51   }
52   cout << ";" << endl;
53  }
54
55  for (int n = 0, count=0;n < coluna;n++) {
```

```
56    if (matrix.at(n).at(n) == 0)
57     count++;
58     if (count == coluna)
59      cout << "\n" << "Exames importados corretamente \n . \n . \n . \n";
60   }
61
62   myReadFile >> clusterx;
63   cluster.resize(clusterx);
64   myReadFile >> clustery;
65
66
67   cout << "\n \n \n \n \n A CARREGAR CLUSTERS \n \n \n \n \n" << endl;
68
69   for (int j = 0; j < clusterx;j++) {
70    for (int k = 0; k < clustery;k++) {
71     myReadFile >> number;
72     if(number==1)
73     cluster.at(j).push_back(k);
74    }
75   }
76
77   for (int i = 0; i < clusterx; i++) {
78
79    cout << " cluster " << i+1 << ":";
80    for (int j = 0; j < cluster.at(i).size(); j++) {
81     cout << " " << cluster.at(i).at(j) << " ";
82     // PARA OBSERVAR
83    }
84    cout << ";" << endl;
85   }
86
87
88   cout << "\n \n \n \n \n Clusters importados \n \n \n \n \n" << endl;
89
90   cout << " A importar nomes dos exames\n\n\n";
91   int dim_exam;
92   myReadFile >> dim_exam;
93   string aux;
94   for (int count = 0; count < dim_exam; count++) {
95
96    //getline(myReadFile, aux);
97    myReadFile >> aux;
98    name.push_back(aux);
99    //cout << name.at(count) << endl;
100  }
101
102  for (int count = 0; count < name.size(); count++) {
103
104   cout << name.at(count) << endl;
105
106  }
107
108  cout << "\n \n \n \n \n Nomes de exames importados \n \n \n \n \n" << endl;
109
110  cout << "\n \n \n \n \n Importar clusters de anos curriculares \n \n \n \n \n" << endl;
111
112
```

```
113   int size_second_cluster;
114   int exame;
115   myReadFile >> size_second_cluster;
116   cluster2.resize(size_second_cluster);
117   int size_aux;
118   for (int h = 0; h < size_second_cluster; h++) {
119    myReadFile >> size_aux;
120    for (int g = 0; g < size_aux;g++) {
121     myReadFile >> exame;
122     cluster2.at(h).push_back(exame);
123    }
124   }
125
126
127   for (int f = 0; f < cluster2.size(); f++) {
128    cout << "Exames " << f + 1 << " ano: ";
129    for (int d = 0; d < cluster2.at(f).size();d++) {
130     cout << cluster2.at(f).at(d) << " ";
131    }
132    cout << ".\n";
133   }
134   cout << "\n \n \n \n \n clusters importandos \n \n \n \n \n" << endl;
135
136   myReadFile.close();
137
138 }
139
140
141 InstanceData::~InstanceData()
142 {
143 }
```

A.3 Descodificador

```
1  #include "SampleDecoder.h"
2  #include "instancedata.h"
3  #include <iostream>
4  #include <cmath>
5  #include <vector>
6
7  bool pairCompare(const std::pair<int, int>& firstElem, const std::pair<int, int>&
       secondElem) {
8   return firstElem.second < secondElem.second;
9
10 }
11 const int weekend_account( int period) {
12  int retorno = period;
13  int account = 0;
14  while (1) {
15   retorno -= 15;
16   if (retorno > 0)
17    account++;
18   else break;
19  }
20  return period + account*6;
21 }
22
23 SampleDecoder::SampleDecoder(const InstanceData &_instance){
24  instance = &_instance;
25 }
26
27 SampleDecoder::~SampleDecoder(){ }
28
29 // Runs in \Theta(n \log n):
30 double SampleDecoder::decode(const std::vector< double >& chromosome) const {
31  int sem = 5; // semanas
32  int dist_n = 11; // Parametro de espacamento consideravel para e'poca normal
33  int dist_n_max = 15; // Parametro de espacamento maximo consideravel para e'poca normal
34  int dist_r = 8; // Parametro de espacamento consideravel para e'poca recurso
35  int dist_r_max = 11; // Parametro de espacamento maximo consideravel para e'poca
       recurso
36  int dist_n_r = 8; // Parametro de espacamento minimo entre e'poca normal e e'poca de
       recurso
37  int dist_n_r_max = 11; // Parametro de espacamento maximo entre e'poca normal e e'poca
       de recurso
38  int threshold = 0; // Numero de alunos maximo que se pode ignorar para que exista mais
       que 1 exame por dia.
39  std::vector<vector<int>> time(sem*5*3);
40
41  float escala = (float)time.size();
42
43  int pos;
44  // criar vetor de tempos
45  for (int i = 0; i < chromosome.size();i++) {
46   pos = ceil(chromosome.at(i)*escala); // ceil para fazer round ao proximo inteiro evita
         problemas para o periodo 1 e periodo 60
47   time.at(pos-1).push_back((int)(i/2));
48   i++;
```

```
49    pos = ceil(chromosome.at(i)*escala); // ceil para fazer round ao proximo inteiro evita
          problemas para o periodo 1 e periodo 60
50    time.at(pos - 1).push_back((int)(i / 2));
51   }
52
53   // LEITURA DO VETOR DE TEMPOS
54
55   /*for (int i = 0; i < time.size();i++) {
56    cout << i+1 << " - ";
57    for (int j = 0; j < time.at(i).size();j++) {
58     cout << " " << time.at(i).at(j); //o +1 e' porque os valores estao com as posicoes
          dos vetores, nao comecam em 1 mas em 0
59       }
60    cout << endl;
61   }*/
62
63   //*************************************************************************
64
65   // Restricao forte 1 - Nao pode haver mais que 3 exames por periodo
66
67
68   int custo_1 = 0;
69   int count = 0;
70   for (int k = 0; k < time.size();k++) {
71    if (time.at(k).size() > 3)
72     count++;
73   }
74   if (count != 0)
75    custo_1+= 999;
76
77
78   /*************************************************************************/
79
80   //  Restricao forte 2 - Exames que ocorrem em cada periodo nao podem ter estudantes em
         comum
81
82   float custo_2 = 0; // inicializar o custo_2 a zero
83   size_t size; // valor que da o tamanho de cada vetor na matriz de tempo
84   for (int k = 0; k < time.size();k++) // percorrer todos os vetores da matriz
85   {
86    size = time.at(k).size();
87    int m = 0;
88    if (size > 1) {
89     for (std::vector<int>::iterator it = time.at(k).begin(); m< time.at(k).size()-1;m++,
          it++) // todas as combinacoes entre as posicoes do vetor
90      {
91       std::vector<int>::iterator it_next; // apontador ao objeto seguinte
92       it_next = it;
93       it_next++; // comeca sempre uma posicao a frente do apontador principal
94       int obj = *it;
95       for ( ; it_next != time.at(k).end(); it_next++)
96       {
97        int obj2 = *it_next;
98        if (obj != obj2) {
99         if (instance->getconflito(obj, obj2) != 0)
100         //custo_2=custo_2+ instance->getconflito(obj, obj2);
101         custo_2 += 9999;
```

```
102         }
103         else custo_2 = custo_2 + 99999;
104        }
105       }
106     }
107    }
108
109  /*
       ********************************************************************************
       */
110
111  //  Restricao forte 3 Garantir que nenhum estudante tem exames em periodos adjacentes no
       mesmo dia
112   // + restricao forte 4 - Exames od memso ano curriculkar nao podem ocorrer no mesmo dia
113   // + restricao forte 5 - Exames de anos consecutivos nao podem ocorrer no mesmo dia
114   // + restricao fraca 1 - Cada estudante tem no maximo 1 exame por dia
115   int custo_3 = 0; // restricao forte 3
116   int custo_same_day = 0; // restricao fraca 1
117   int custo_ano = 0; // restricao forte 4
118   int custo_consecutive = 0; // restricao forte 5
119   // VERSAO 3 Compara o periodo do meio com os periodos adjacentes, em cada dia
120   for (int g = 0, count = 0; g < time.size() - 2;) { //percorrer o vetor de tempos
121    vector<int> period_vector2;
122    vector<int> period_vector3;
123    vector<int> period_vector4;
124    int count_1 = 0;
125    int count_2 = 0;
126    int count_3 = 0;
127    for (int t = 0; t < time.at(g).size();t++) {
128     period_vector2.push_back(time.at(g).at(t));
129     instance->getmatch(time.at(g).at(t), &count_1, &count_2, &count_3);
130    }
131    g++;
132    for (int y = 0; y < time.at(g).size();y++) {
133     period_vector3.push_back(time.at(g).at(y));
134     instance->getmatch(time.at(g).at(y), &count_1, &count_2, &count_3);
135    }
136    g++;
137    for (int z = 0; z < time.at(g).size();z++) {
138     period_vector4.push_back(time.at(g).at(z));
139     instance->getmatch(time.at(g).at(z), &count_1, &count_2, &count_3);
140    }
141    if (period_vector3.empty() == 0 && (period_vector2.empty() == 0 || period_vector4.
         empty() == 0)) {
142     for (std::vector<int>::iterator ti = period_vector3.begin(); ti != period_vector3.end
          (); ti++) {
143      std::vector<int>::iterator ti4 = period_vector4.begin();
144      std::vector<int>::iterator ti2 = period_vector2.begin();
145      for (; ti4 != period_vector4.end();ti4++) {
146       if (*ti != *ti4) {
147        custo_3 += instance->getconflito(*ti, *ti4);
148       }
149       else custo_3 += 100;
150      }
151      for (; ti2 != period_vector2.end();ti2++) {
152       if (*ti != *ti2) {
153        custo_3 += instance->getconflito(*ti, *ti2);
```

```
154         }
155         else custo_3 += 100;
156        }
157       }
158      }
159      // VERSAO 3.1 COMPARA O PRIMEIRO PERIODO COM O ULTIMO, mas so se o numero de alunos
             for superior a um dado valor e' que da trigger
160      if (period_vector2.empty() == 0 && period_vector4.empty() == 0) {
161       std::vector<int>::iterator ti4 = period_vector4.begin();
162       std::vector<int>::iterator ti2 = period_vector2.begin();
163       for (;ti2 != period_vector2.end();ti2++) {
164        for (;ti4 != period_vector4.end();ti4++) {
165         if (*ti2 != *ti4) {
166          if (instance->samecluster(*ti2, *ti4) == 0) {
167           if (instance->getconflito(*ti2, *ti4) > threshold)
168
169            custo_same_day += instance->getconflito(*ti2, *ti4);
170          }
171          else custo_ano += 100; // EXAMES DO MESMO ANO NAO PODEM OCORRER NO MESMO DIA
172         }
173         else custo_3 += 100;
174        }
175       }
176      }
177      g++;
178      if (count_1 > 0 && count_2 > 0) {
179       custo_consecutive += 999;
180      }
181      if (count_2 > 0 && count_3 > 0) {
182       custo_consecutive += 999;
183      }
184
185     }
186
187     //*************************************************************************************
188
189     //restricao forte 6 - Para cada conjunto de cadeiras que qualquer estudante tenha, os
            exames de recurso apenas iniciam quando os exames de e'poca normal terminam
190
191     int custo_6 = 0;
192     size_t matrix_size = instance->getsizeofmatrix();
193     for (int j = 0; j < matrix_size;j++) {
194      pair<int, int> normal_times_aux;
195      vector<pair<int, int>> normal_times;
196      pair<int, int> recurso_times_aux;
197      vector<pair<int, int>> recurso_times;
198      vector<int>aux = instance->getlinhadeconflito(j);
199      for (int k = 0;k < aux.size();k++) {
200       if (aux.at(k) != 0) {
201        normal_times_aux = make_pair(k, ceil(chromosome.at(k * 2)*escala));
202        normal_times.push_back(normal_times_aux);
203        recurso_times_aux = make_pair(k, ceil(chromosome.at(k * 2+1)*escala));
204        recurso_times.push_back(recurso_times_aux);
205       }
206      }
207      sort(normal_times.begin(), normal_times.end(), pairCompare);
208      sort(recurso_times.begin(), recurso_times.end(), pairCompare);
```

```
209     int max = normal_times.back().second;
210     vector<pair<int, int>> aux_times;
211     for (int p = 0;p < recurso_times.size();p++) {
212      if (recurso_times.at(p).second <= max)
213       aux_times.push_back(recurso_times.at(p));
214     }
215     for (int t = 0; t < aux_times.size(); t++) {
216      for (std::vector<pair<int, int>>::iterator normal_shift = normal_times.begin();
            normal_shift != normal_times.end();normal_shift++) {
217       if ( (*normal_shift).second > aux_times.at(t).second) {
218        if((*normal_shift).first!= aux_times.at(t).first)
219        custo_6 += instance->getconflito((*normal_shift).first, aux_times.at(t).first);
220        else custo_6 += 200;
221       }
222      }
223     }
224    }
225    //*********************************************************************************
226    float custo_N = 0;
227    float custo_R = 0;
228    float custo_dist_NormalRecurso = 0;
229    if ((custo_1 + custo_2 + custo_6 + custo_3 + custo_ano + custo_consecutive) == 0) {
230     //  restricao fraca 3 - avaliacao do espacamento temporal entre clusters
231     int rectify;
232     size_t size_cluster = instance->getsizeofclustermatrix();
233     //epoca normal
234     for (int i = 0; i < size_cluster;i++) {
235      vector<int> cluster = instance->getcluster(i);
236      vector<int> pos;
237      for (int j = 0; j < cluster.size();j++) {
238       rectify = weekend_account(ceil(chromosome.at(cluster.at(j) * 2)* escala));
239       pos.push_back(rectify); // x2 por cada par de genes corresponde a um unico exame
240      }
241      sort(pos.begin(), pos.end());
242      for (int k = 0; k < cluster.size() - 1;k++) {
243       if ((pos.at(k + 1) - pos.at(k)) < dist_n || (pos.at(k + 1) - pos.at(k)) > dist_n_max
             )
244        custo_N += 100;
245       else custo_N -= 100;
246      }
247     }
248
249     //epoca recurso
250     for (int i = 0; i < size_cluster;i++) {
251      vector<int> cluster = instance->getcluster(i);
252      vector<int> pos;
253      for (int j = 0; j < cluster.size();j++) {
254       rectify = weekend_account(ceil(chromosome.at(cluster.at(j) * 2 + 1)* escala));
255       pos.push_back(rectify); // +1 para aceder ao gene correspondente ao recurso
256      }
257      sort(pos.begin(), pos.end());
258      for (int k = 0; k < cluster.size() - 1;k++) {
259       if ((pos.at(k + 1) - pos.at(k)) < dist_r || (pos.at(k + 1) - pos.at(k)) > dist_r_max
             )
260        custo_R += 100;
261       else custo_R -= 100;
262      }
```

```
263     }
264
265     // restricao fraca 2 - Espacamentos entre e'pocas normal e recurso
266     vector<int>aux_times_normal;
267     vector<int>aux_times_recurso;
268     for (int j = 0; j < matrix_size;j++) {
269      vector<int>aux = instance->getlinhadeconflito(j);
270      for (int k = 0;k < aux.size();k++) {
271       if (aux.at(k) != 0) {
272        rectify = weekend_account(ceil(chromosome.at(k * 2)*escala));
273        aux_times_normal.push_back(rectify);
274        rectify = weekend_account(ceil(chromosome.at(k * 2 + 1)*escala));
275        aux_times_recurso.push_back(rectify);
276       }
277      }
278      sort(aux_times_normal.begin(), aux_times_normal.end());
279      sort(aux_times_recurso.begin(), aux_times_recurso.end());
280      if ((aux_times_recurso.front() - aux_times_normal.back()) < dist_n_r || (
             aux_times_recurso.front() - aux_times_normal.back()) > dist_n_r_max)
281       custo_dist_NormalRecurso += 100;
282      else custo_dist_NormalRecurso -= 100;
283     }
284    }
285   // RETORNO DO FITNESS
286     if ((custo_1 + custo_2 + custo_6 + custo_3 + custo_ano + custo_consecutive) != 0)
287      return custo_1 + custo_2 + 5 * custo_6 + 10 * custo_3 + 5*custo_ano +
             custo_consecutive;
288    else {
289    //cout << "zero zero" << endl;
290     return -100000 + custo_dist_NormalRecurso + 10 * custo_N + 5 * custo_R +
           custo_same_day;
291    }
292   }
```

A.4 Alocação em salas

A.4.1 Leitura de dados

```
1
2  vector<pair<string, int>> salas_escrito;
3   vector<pair<string, int>> salas_computadores;
4   vector<int> ecomp;
5   ifstream salas;
6
7   salas.open("src/salas.txt");
8   if (salas.fail()) {
9    cout << "something wrong with input file: " << "salas.txt" << endl;
10   cout << "pgm will terminate now " << endl;
11   exit(0);
12  }
13
14  string type;
15  int value;
16  int aux;
17  int loops;
18  salas >> loops;
19
20  while (loops != 0) {
21
22   salas >> type;
23
24   if (type == "ESCRITO") {
25    salas >> value;
26    int size_table = value;
27    for (int i = 0; i < size_table;i++) {
28     salas >> type;
29     salas >> aux;
30     salas_escrito.push_back(make_pair(type, aux));
31    }
32    sort(salas_escrito.begin(), salas_escrito.end(), pairCompare3);
33   }
34
35   if (type == "COMP") {
36    salas >> value;
37    int size_table_2 = value;
38    for (int i = 0; i < size_table_2;i++) {
39     salas >> type;
40     salas >> value;
41     salas_computadores.push_back(make_pair(type, value));
42    }
43    sort(salas_computadores.begin(), salas_computadores.end(), pairCompare3);
44   }
45
46   loops--;
47  }
48
49
50  salas >> type;
51
52  if (type == "ECOMP") {
```

```
53    salas >> value;
54    size_t size_ecomp = value;
55    for (int i = 0; i < size_ecomp;i++) {
56     salas >> value;
57     ecomp.push_back(value);
58    }
59
60   }
61
62   salas >> value;
63   vector<int> n_estudantes;
64   int  number;
65   for (int j = 0; j < value;j++) {
66    salas >> number;
67    n_estudantes.push_back(number);
68   }
```

A.4.2 Algoritmo

```
1
2  vector<vector<pair<int, vector<string>>>> time_rooms(sem*5*3);
3  //Cada periodo, cada exame nesse periodo, o numero do exame e o vetor com as salas que
       utilizou
4   for (int j = 0; j < time.size();j++) {
5    vector<pair<int, int>> e_cap;
6    pair<vector<string>,vector<int>> virtual_rooms_escrito; // int 1 ou 0 representa a
       ocupacao da sala
7    pair<vector<string>, vector<int>> virtual_rooms_comp;
8    for (int g = 0;g < salas_escrito.size();g++) {
9     virtual_rooms_escrito.first.push_back(salas_escrito.at(g).first);
10    virtual_rooms_escrito.second.push_back(0);
11   }
12   for (int g = 0;g < salas_computadores.size();g++) {
13    virtual_rooms_comp.first.push_back(salas_computadores.at(g).first);
14    virtual_rooms_comp.second.push_back(0);
15   }
16   int side;
17   if (time.at(j).size() != 0) {
18    for (int i = 0; i < time.at(j).size();i++) {
19     e_cap.push_back(make_pair(time.at(j).at(i),n_estudantes.at(time.at(j).at(i))));
20    }
21    sort(e_cap.begin(), e_cap.end(), pairCompare4);
22    for (std::vector<pair<int, int>>::iterator it = e_cap.begin();it != e_cap.end();it++)
        {
23     vector<string> aux;
24     side = 0;
25     for (int i = 0;i < ecomp.size();i++) {
26      if ((*it).first == ecomp.at(i))
27       side = 1;
28     }
29     if (side == 0) // exame escrito
30     {
31      int capacidade = (*it).second;
32      for (int k = 0;k < salas_escrito.size() && capacidade>0;k++) {
33       if ( virtual_rooms_escrito.second.at(k)==0) {
34        capacidade -= salas_escrito.at(k).second;
35        aux.push_back(salas_escrito.at(k).first);
```

```
36          virtual_rooms_escrito.second.at(k) = 1;
37         }
38        }
39        if (capacidade > 0)
40         cout << "Nao ha espaco suficiente para alocar alunos de " << Data.getnameofexam((*
               it).first) << endl;
41        else time_rooms.at(j).push_back(make_pair((*it).first, aux));
42       }
43       if (side == 1) { // exame computador
44        int capacidade = (*it).second;
45        for (int k = 0;k < salas_computadores.size() && capacidade > 0;k++) {
46         if (virtual_rooms_comp.second.at(k) == 0) {
47          capacidade -= salas_computadores.at(k).second;
48          aux.push_back(salas_computadores.at(k).first);
49          virtual_rooms_comp.second.at(k) = 1;
50         }
51        }
52        if (capacidade > 0)
53         cout << "Nao ha espaco suficiente para alocar alunos de " << Data.getnameofexam((*
               it).first) << endl;
54        else time_rooms.at(j).push_back(make_pair((*it).first, aux));
55       }
56      }
57     }
58     else continue;
59    }
60
61    ofstream room_agenda;
62
63    room_agenda.open("agenda.csv");
64
65    room_agenda << "Periodo, Exame, Salas\n";
66
67    //Escrever para o terminal e para um ficheiro, a leitura do time_rooms
68    for (int i = 0; i < time_rooms.size();i++) {
69     cout << "PERIODO "<<i<<"\n\n";
70     for (int j = 0;j < time_rooms.at(i).size();j++) {
71      room_agenda << i;
72      cout << " Exame de " << Data.getnameofexam(time_rooms.at(i).at(j).first) << " alocado
            nas salas: ";
73      room_agenda << "," << Data.getnameofexam(time_rooms.at(i).at(j).first) << ",";
74      for (int z = 0; z < time_rooms.at(i).at(j).second.size();z++) {
75       room_agenda << time_rooms.at(i).at(j).second.at(z) << ";";
76       cout << time_rooms.at(i).at(j).second.at(z) << " ";
77      }
78      cout << endl;
79      room_agenda << "\n";
80     }
81    }
82    room_agenda.close();
```

Anexo B

Ficheiros e Dados

B.1 Ficheiro diagnóstico

Diagnóstico efetuado em 1497990877
Semente utilizada: 8847
Populacoes utilizadas: 2
Tamanho da populaçao: 100
Threads utilizadas: 2
Fraçao da populaçao a ser elite: 0.2
Fraçao mutantes: 0.15
Probabilidade offspring: 0.7
Troca de cromossomas elite a todas as : 100 geracoes
Número de cromossomas elite a trocar: 2
A solucao otima foi obtida em: 2459
As restricoes fortes foram cumpridas a partir de: 312
Melhor soluçao: 25 47 12 63 21 58 9 73 31 51 22 59 3 61 16 44 10 56 27 47 28 57 10 46 27 46 7 55 28 49 30 70 34 58 18 40 4 45 24 69 15 61 4 56 13 45 22 37 1 47 21 49 39 49 24 56 2 64 13 52 22 60 33 34 32 61 14 69 21 50 2 37
Com fitness de: -134885
FISI e COPT tem os seus exames no mesmo dia.
SINF e SBIC tem os seus exames no mesmo dia.
SEEN e STEL tem os seus exames no mesmo dia.
IOPE e EGES tem os seus exames no mesmo dia.
DOIC e CIRC tem os seus exames no mesmo dia.
FISI e ELEC3 tem os seus exames no mesmo dia.
Espaçamento temporal nos clusters durante epoca normal:
Cluster 0- Os espacamentos sao: 5 1.33333 dias.
Cluster 1- Os espacamentos sao: 5 4 dias.
Cluster 2- Os espacamentos sao: 3.66667 4.66667 5 dias.
Cluster 3- Os espacamentos sao: 3.66667 5 4.33333 1 dias.
Cluster 4- Os espacamentos sao: 3.66667 5 4.33333 2.66667 dias.
Cluster 5- Os espacamentos sao: 4 3.66667 4 dias.
Cluster 6- Os espacamentos sao: 4 4 3.66667 dias.
Cluster 7- Os espacamentos sao: 5 3.66667 4 dias.
Cluster 8- Os espacamentos sao: 4 3.66667 dias.
Cluster 9- Os espacamentos sao: 4 5 dias.
Cluster 10- Os espacamentos sao: 7 dias.

Cluster 11- Os espacamentos sao: 3.66667 4.66667 dias.
Cluster 12- Os espacamentos sao: 3.66667 4.66667 dias.
Cluster 13- Os espacamentos sao: 3.66667 5 dias.
RECURSOS
Cluster 0- Os espacamentos sao: 3.66667 3.66667 dias.
Cluster 1- Os espacamentos sao: 14 1 dias.
Cluster 2- Os espacamentos sao: 2.66667 2.66667 2.66667 dias.
Cluster 3- Os espacamentos sao: 1.66667 0.666667 3 2.66667 dias.
Cluster 4- Os espacamentos sao: 3.33333 0.666667 1.66667 3.66667 dias.
Cluster 5- Os espacamentos sao: 3.33333 3 3 dias.
Cluster 6- Os espacamentos sao: 3.33333 2.66667 3.66667 dias.
Cluster 7- Os espacamentos sao: 3.33333 2.66667 3 dias.
Cluster 8- Os espacamentos sao: 3.33333 3.66667 dias.
Cluster 9- Os espacamentos sao: 2.66667 2.66667 dias.
Cluster 10- Os espacamentos sao: 3.33333 dias.
Cluster 11- Os espacamentos sao: 2.66667 3.66667 dias.
Cluster 12- Os espacamentos sao: 2.66667 3.33333 dias.
Cluster 13- Os espacamentos sao: 2.66667 3.33333 dias.
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

B.2 Dados do problema

36 //número de linhas e coluna da matriz. A matriz corresponde ao número de estudantes em comum entre unidades curriculares

0 263 272 36 17 0 1 0 0 0 0 0 0 0 1 33 9 45 6 11 14 11 1 0 0 0 5 5 1 1 2 0 0 1 0 0 263 0 271 28 8 1 2 1 1 1 1 1 0 0 0 25 4 36 1 1 5 2 3 0 0 1 2 3 0 0 0 1 0 1 0 2 272 271 0 24 6 0 0 0 0 0 0 0 0 0 0 0 25 0 42 2 1 5 1 0 0 0 0 1 4 2 2 2 0 0 1 0 0 36 28 24 0 40 7 7 2 0 0 1 1 0 0 1 203 16 202 26 21 34 19 14 1 3 3 20 24 4 7 9 2 0 13 1 3 17 8 6 40 0 41 42 1 0 0 1 1 0 0 0 35 76 38 32 77 106 75 5 3 3 1 26 24 6 5 4 4 0 46 0 1 0 1 0 7 41 0 45 2 0 0 0 0 0 1 2 7 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 5 0 47 0 0 1 2 0 7 42 45 0 13 3 4 2 2 2 3 2 5 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 10 0 53 0 0 0 1 0 2 1 2 13 0 6 7 12 12 3 5 14 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 34 0 13 0 0 0 1 0 0 0 0 3 6 0 8 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 6 0 2 0 0 0 1 0 0 0 0 4 7 8 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 7 0 3 0 0 0 1 0 1 1 0 2 12 0 0 0 13 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 8 0 2 0 0 0 1 0 1 1 0 2 12 0 0 13 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 9 0 5 0 0 0 0 0 0 0 0 2 3 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 1 0 0 0 0 0 0 0 1 3 5 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 4 0 3 0 0 1 0 0 1 0 2 2 14 0 0 0 0 0 0 0 1 4 2 0 4 1 8 28 0 0 0 0 0 1 0 0 14 1 3 0 0 33 25 25 203 35 7 5 0 1 1 0 0 0 0 1 0 11 184 24 12 29 10 6 1 1 1 17 23 2 6 7 3 1 10 0 3 9 4 0 16 76 0 0 0 0 0 0 0 0 0 4 11 0 12 0 88 82 86 18 5 4 1 0 0 0 0 0 0 0 0 0 2 2 45 36 42 202 38 7 7 1 0 0 0 0 0 0 2 184 12 0 11 18 31 15 5 0 0 0 9 7 1 1 1 1 0 11 1 1 6 1 2 26 32 0 0 0 0 0 0 0 0 0 0 24 0 11 0 0 32 0 0 0 0 0 39 47 15 18 22 0 2 0 0 0 11 1 1 21 77 0 0 0 0 0 0 0 0 0 4 12 88 18 0 0 84 87 20 5 4 3 0 0 0 0 0 0 0 0 1 4 14 5 5 34 106 0 0 0 0 0 0 0 0 0 1 29 82 31 32 84 0 81 5 1 2 0 28 29 3 4 2 0 0 0 1 1 11 2 1 19 75 0 0 0 0 0 0 0 0 0 8 10 86 15 0 87 81 0 27 8 8 2 0 0 0 0 0 0 0 2 5 1 3 0 14 5 0 0 0 0 0 0 0 0 0 28 6 18 5 0 20 5 27 0 21 22 36 0 0 0 0 0 0 0 0 6 40 0 0 0 1 3 0 0 1 0 0 0 0 0 1 0 1 5 0 0 5 1 8 21 0 22 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 0 0 0 0 0 0 0 0 0 0 1 4 0 0 4 2 8 22 22 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 3 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 3 0 2 36 0 0 0 0 0 0 0 0 0 0 0 0 0 38 5 2 1 20 26 0 0 0 0 0 0 0 0 0 0 17 0 9 39 0 28 0 0 0 0 0 0 47 14 11 12 0 0 0 0 0 5 3 4 24 24 0 0 0 0 0 0 0 0 0 0 0 23 0 7 47 0 29 0 0 0 0 0 47 0 22 23 27 0 3 0 0 0 1 0 2 4 6 0 0 0 0 0 0 0 0 0 1 2 0 1 15 0 3 0 0 0 0 0 14 22 0 37 41 0 2 0 0 0 1 0 2 7 5 0 0 0 0 0 0 0 0 0 0 0 6 0 1 18 0 4 0 0 0 0 0 11 23 37 0 44 0 2 0 0 0 2 0 2 9 4 0 0 0 0 0 0 0 0 0 0 0 7 0 1 22 0 0 2 0 0 0 0 0 12 27 41 44 0 0 3 0 0 0 0 1 0 2 4 5 10 34 6 7 8 9 3 4 14 3 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 13 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 2 0 0 0 0 0 0 0 0 3 2 2 3 0 0 0 0 0 1 1 1 13 46 47 53 13 2 3 2 5 1 3 3 10 0 11 0 0 0 0 0 0 0 0 0 0 0 0 0 13 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 2 1 0 1 1 2 6 0 0 0 0 0 0 0 0 0 0 0 3 0 2 0 3 1 0 0 0 0 0 0 0 0 0 0 3 2 1 0 4 1 5 40 0 0 38 0 0 0 0 0 0 0 0 3 0

14 36 // Linhas e colunas da matriz de clusters de exames, 1 para UCs pertencentes ao cluster.

1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0

36 // número de nomes de UC a ler

AMAT2 FISI CIRC ELEC1 EGES PDSI ELEC3 STEL COPT APRO COMO PGRE SMUL PCIM PCVL TCON EIND SEEN MELE IELE SINF CDIG SBIC SADE SQFI RIND RTDI RESEE MQUA PLIC DOIC IOPE TAT SO ACCM APS

3 // Número de clusters de unidades curriculares

3 0 1 2 // 1º valor -> número de UCs, restantes valores correspondem ao índice do exame

3 15 17 3 // igual

11 21 4 16 19 20 33 6 5 18 27 26 // igual

B.3 Dados das salas e número de estudantes por UC

2 // número de tipos de salas a ler

ESCRITO // identificador do tipo de sala escrito

8 // número de salas

B116 69 // ID da sala e capacidade

B120 69

B215 82

B221 89

B227 89

B231 69

B232C 44

B338 69

COMP // tipo de sala computadores

3 // número de salas

B104 69

B208 52

B213 52

ECOMP // exames que necessitam de computador

3 // número de exames

16 20 35 // índice dos exames

36 // número de UCs para ler o numero de estudantes em cada. 319 310 310 275 200 65 70 51 22 15 17 25 15 24 45 268 105 272 87 122 153 120 123 57 57 54 64 87 78 56 71 57 77 73 13 57

B.4 Agenda das salas exemplo

Periodo	Exame	Salas					
0	EIND	B104;B208;	APRO	B215;	47	PCVL	B227;
0	PGRE	B221;	AMAT2	B221;B227;B215;B116;	48	SEEN	B221;B227;B215;B116;
0	COPT	B227;	ELEC1	B221;B227;B215;B116;	49	SQFI	B221;
1	CIRC	B221;B227;B215;B116;	IOPE	B221;	51	CDIG	B221;B227;
7	TAT	B221;	EGES	B221;B227;B215;	52	RESEE	B221;
8	SINF	B104;B208;B213;	DOIC	B221;	55	STEL	B221;
8	PDSI	B221;	RIND	B227;	56	TCON	B221;B227;B215;B116;
9	SADE	B221;	ACCM	B215;	57	SBIC	B221;B227;
10	SEEN	B221;B227;B215;B116;	TCON	B221;B227;B215;B116;	57	PDSI	B215;
12	SBIC	B221;B227;	SMUL	B221;	58	RTDI	B221;
12	MQUA	B215;	CDIG	B221;B227;	59	EIND	B104;B208;
12	STEL	B116;	ELEC3	B215;	59	COPT	B221;
14	FISI	B221;B227;B215;B116;	DOIC	B221;	60	EGES	B221;B227;B215;
15	IELE	B221;B227;	IELE	B221;B227;	60	TAT	B116;B120;
15	RESEE	B215;B116;	ELEC1	B221;B227;B215;B116;	60	SMUL	B231;
16	SO	B221;	MELE	B221;	62	APS	B104;
17	PLIC	B221;	FISI	B221;B227;B215;B116;	62	APRO	B221;
18	RTDI	B221;	MQUA	B120;B231;	63	COMO	B221;
18	APS	B104;	SADE	B338;	64	AMAT2	B221;B227;B215;B116;
18	COMO	B227;	PGRE	B221;	68	ELEC3	B221;
20	SQFI	B221;	CIRC	B221;B227;B215;B116;	69	SINF	B104;B208;B213;
20	PCVL	B227;	PCIM	B120;	71	IOPE	B221;
21	MELE	B221;	ACCM	B231;	73	SO	B221;
21	PCIM	B227;	RIND	B221;	74	PLIC	B221;

Figura B.1: Resultado exemplo de alocação das salas

Anexo C

Calendário exemplo

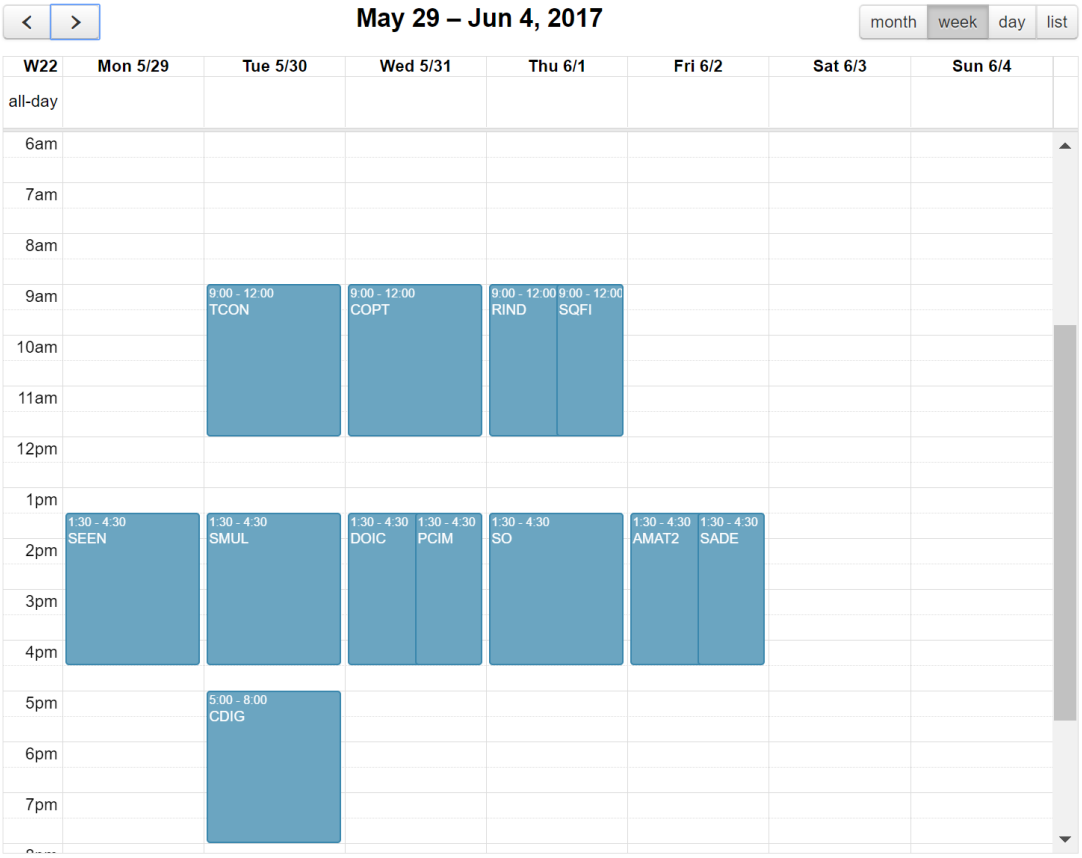


Figura C.1: 1ª semana de um calendário exemplo

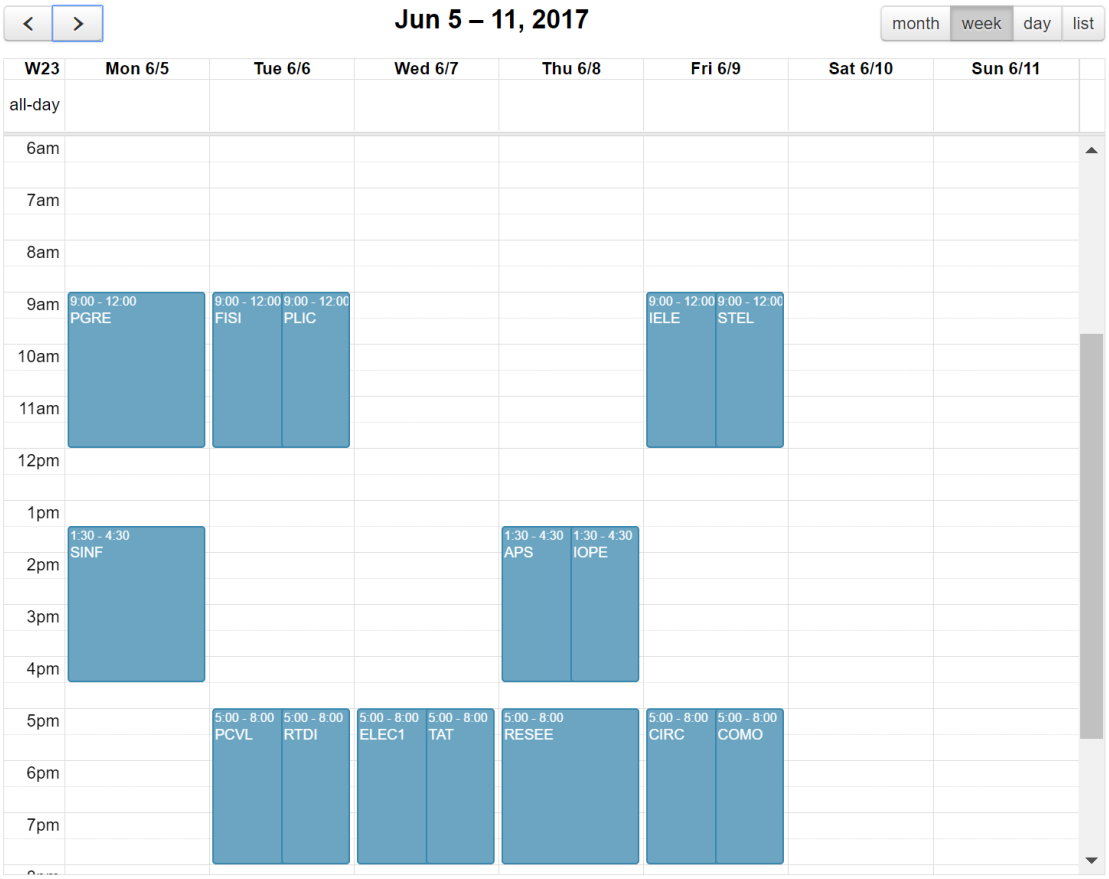


Figura C.2: 2ª semana de um calendário exemplo

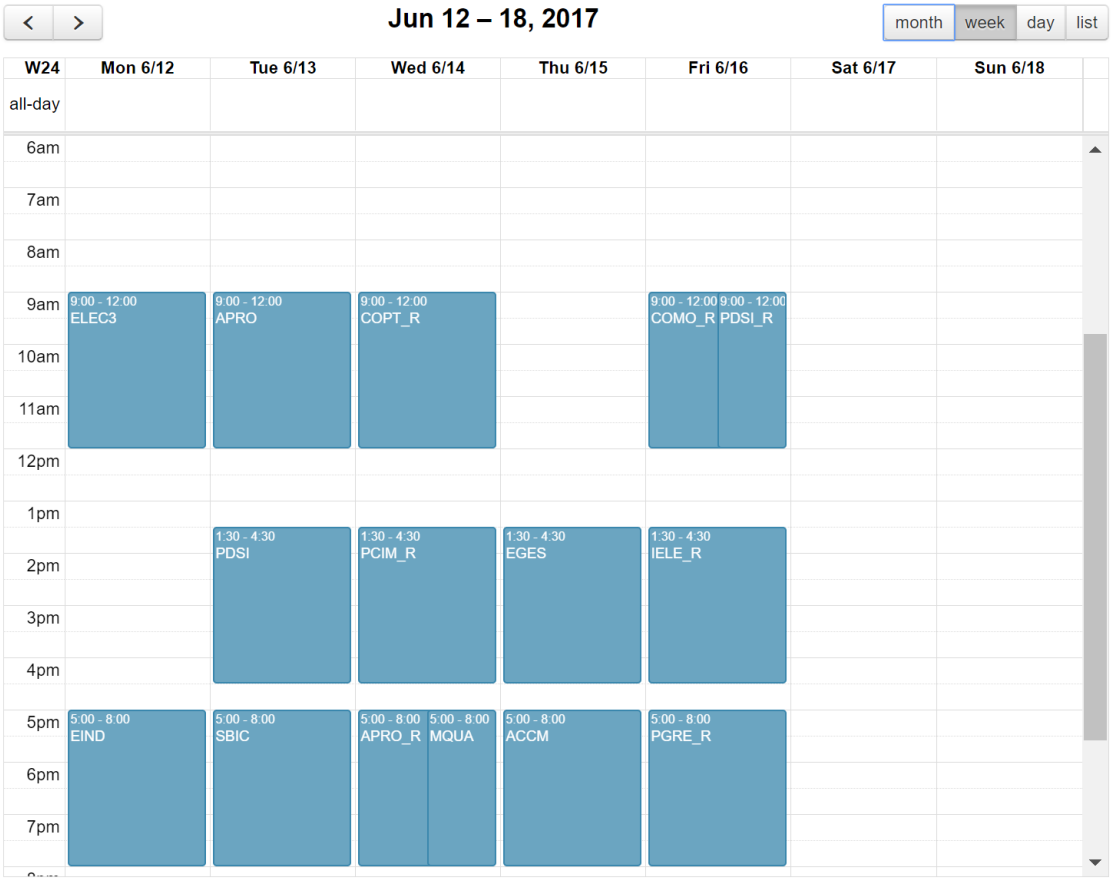


Figura C.3: 3ª semana de um calendário exemplo

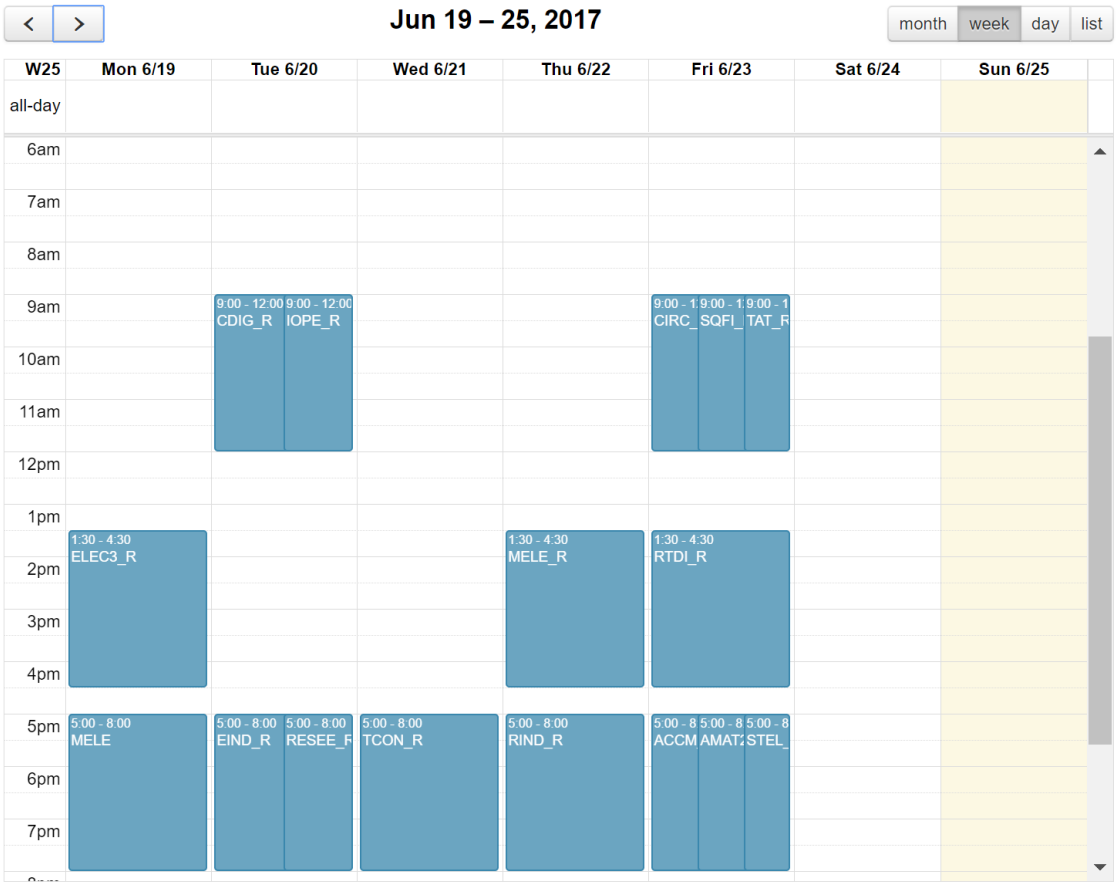


Figura C.4: 4ª semana de um calendário exemplo

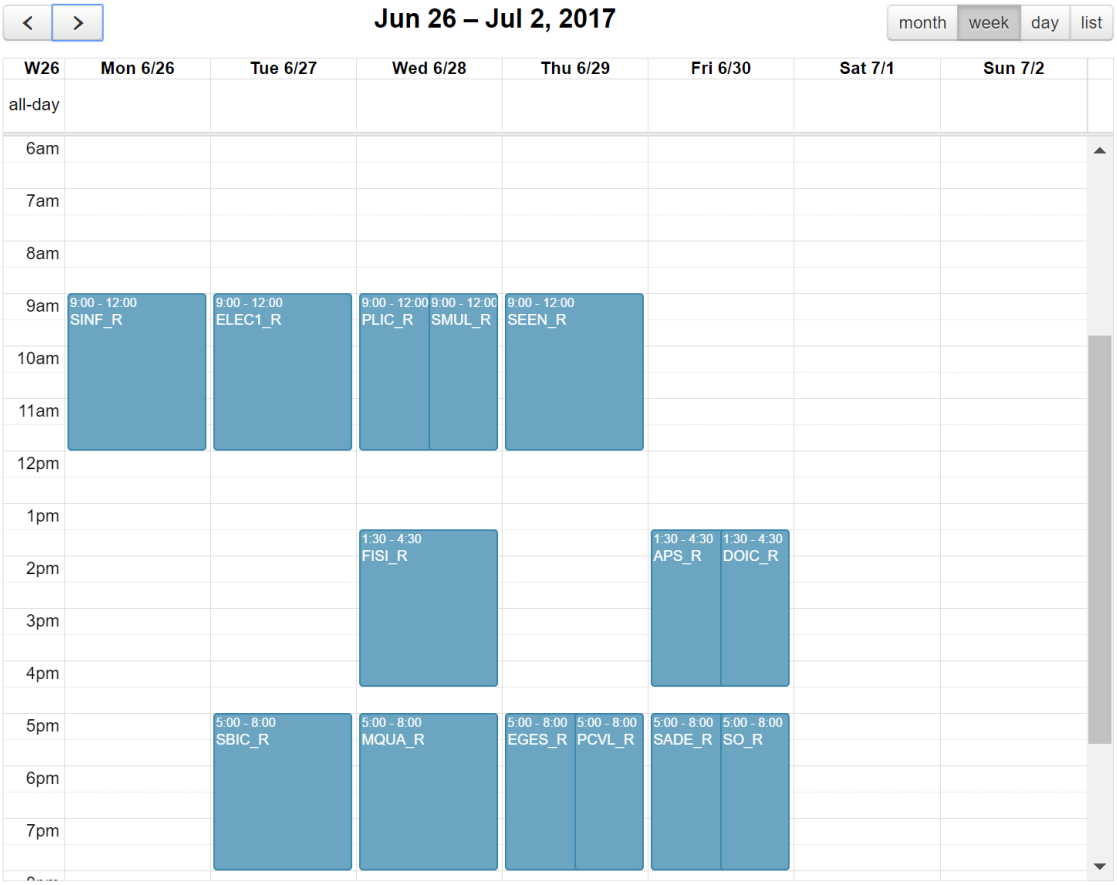


Figura C.5: 5ª semana de um calendário exemplo

Referências

[1] Anthony Wren. Scheduling, timetabling and rostering - a special relationship? Em *Selected Papers from the First International Conference on Practice and Theory of Automated Timetabling*, páginas 46–75, London, UK, UK, 1996. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=646429.692751.

[2] Stephen A. Cook. The complexity of theorem-proving procedures. Em *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, páginas 151–158, New York, NY, USA, 1971. ACM. URL: http://doi.acm.org/10.1145/800157.805047, doi:10.1145/800157.805047.

[3] Richard M. Karp. Reducibility among Combinatorial Problems. Em Raymond E. Miller, James W. Thatcher, e Jean D. Bohlinger, editores, *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department*, páginas 85–103. Springer US, Boston, MA, 1972. DOI: 10.1007/978-1-4684-2001-2_9. URL: http://dx.doi.org/10.1007/978-1-4684-2001-2_9.

[4] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533 – 549, 1986. URL: http://www.sciencedirect.com/science/article/pii/0305054886900481, doi:http://dx.doi.org/10.1016/0305-0548(86)90048-1.

[5] Michel Gendreau e Jean-Yves Potvin, editores. *Handbook of Metaheuristics*, volume 146 de *International Series in Operations Research & Management Science*. Springer US, Boston, MA, 2010. DOI: 10.1007/978-1-4419-1665-5. URL: http://link.springer.com/10.1007/978-1-4419-1665-5.

[6] Luca Di Gaspero e Andrea Schaerf. Tabu search techniques for examination timetabling. Em *International Conference on the Practice and Theory of Automated Timetabling*, páginas 104–117. Springer, 2000. URL: http://link.springer.com/chapter/10.1007/3-540-44629-X_7.

[7] Luca Di Gaspero. Recolour, shake and kick: A recipe for the examination timetabling problem. Em *Proceedings of the fourth international conference on the practice and theory of automated timetabling, Gent, Belgium*, páginas 404–407, 2002. URL: https://www.researchgate.net/profile/Luca_Di_Gaspero/publication/215777265_Recolour_Shake_and_Kick_a_recipe_for_the_Examination_Timetabling_Problem/links/0912f50ae40c393766000000.pdf.

[8] Michel Gendreau, Alain Hertz, e Gilbert Laporte. A tabu search heuristic for the vehicle routing problem. *Management science*, 40(10):1276–1290, 1994. URL: http://pubsonline.informs.org/doi/abs/10.1287/mnsc.40.10.1276.

[9] Scott Kirkpatrick, C. Daniel Gelatt, Mario P. Vecchi, e others. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983. URL: http://leonidzhukov.net/hse/2013/stochmod/papers/KirkpatrickGelattVecchi83.pdf.

[10] Vladimír Cerny. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications*, 45(1):41–51, 1985. URL: http://link.springer.com/article/10.1007/BF00940812.

[11] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, e Edward Teller. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21(6):1087–1092, Junho 1953. URL: http://aip.scitation.org/doi/10.1063/1.1699114, doi:10.1063/1.1699114.

[12] Nader Chmait e Khalil Challita. Using simulated annealing and ant-colony optimization algorithms to solve the scheduling problem. *Computer Science and Information Technology*, 1(3):208–224, 2013. URL: http://www.hrpub.org/journals/article_info.php?aid=671.

[13] Jonathan M. Thompson e Kathryn A. Dowsland. A robust simulated annealing based examination timetabling system. *Computers & Operations Research*, 25(7):637–648, 1998. URL: http://www.sciencedirect.com/science/article/pii/S0305054897001019.

[14] R. Qu, E. K. Burke, B. McCollum, L. T. G. Merlot, e S. Y. Lee. A survey of search methodologies and automated system development for examination timetabling. *Journal of Scheduling*, 12(1):55–89, Fevereiro 2009. URL: http://link.springer.com/10.1007/s10951-008-0077-5, doi:10.1007/s10951-008-0077-5.

[15] Liam TG Merlot, Natashia Boland, Barry D. Hughes, e Peter J. Stuckey. A hybrid algorithm for the examination timetabling problem. Em *International Conference on the Practice and Theory of Automated Timetabling*, páginas 207–231. Springer, 2002. URL: http://link.springer.com/10.1007%2F978-3-540-45157-0_14.

[16] E. Poupaert e Yves Deville. Simulated annealing with estimated temperature. *AI Commun.*, 13(1):19–26, 2000. URL: http://content.iospress.com/articles/ai-communications/aic196.

[17] M. Lundy e A. Mees. Convergence of an annealing algorithm. *Mathematical Programming*, 34(1):111–124, 1986. URL: http://dx.doi.org/10.1007/BF01582166, doi:10.1007/BF01582166.

[18] Graham Kendall. Artificial intelligence: Simulated annealing, introduction. *course run at the The University of Nottingham within the School of Computer Science and IT*, Junho 2002. URL: http://www.cs.nott.ac.uk/~pszgxk/aim/.

[19] Marco Dorigo e Thomas Stützle. *The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances*, páginas 250–285. Springer US, Boston, MA, 2003. URL: http://dx.doi.org/10.1007/0-306-48056-5_9, doi:10.1007/0-306-48056-5_.

[20] Marco Dorigo e Thomas Stützle. The ant colony optimization metaheuristic: Algorithms, applications, and advances. Em *Handbook of metaheuristics*, páginas 250–285. Springer, 2003. URL: http://link.springer.com/chapter/10.1007/0-306-48056-5_9.

[21] M. Duran Toksari. A hybrid algorithm of Ant Colony Optimization (ACO) and Iterated Local Search (ILS) for estimating electricity domestic consumption: Case of Turkey. *International Journal of Electrical Power & Energy Systems*, 78:776–782, Junho 2016. URL: http://linkinghub.elsevier.com/retrieve/pii/S0142061515005840, doi:10.1016/j.ijepes.2015.12.032.

[22] Vittorio Maniezzo, Luca Maria Gambardella, e Fabio de Luigi. *Ant Colony Optimization*, páginas 101–121. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. URL: http://dx.doi.org/10.1007/978-3-540-39930-8_5, doi:10.1007/978-3-540-39930-8_5.

[23] Marco Dorigo e Luca Maria Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation*, 1(1):53–66, 1997. URL: http://ieeexplore.ieee.org/abstract/document/585892/.

[24] Fred Glover e Gary A Kochenberger. *Handbook of metaheuristics*. Kluwer Academic Publishers, Boston, 2003. OCLC: 54042216. URL: http://public.eblib.com/choice/publicfullrecord.aspx?p=197683.

[25] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992.

[26] Mr. Lokesh Kumar Manoj Kr. Mahto. Exam Time Table Scheduling using Genetic Algorithm. *International Journal of Enhanced Research in Management & Computer Applications Vol.4 Issue 8*, 2015.

[27] Marko Čupić, Marin Golub, e Domagoj Jakobović. Exam timetabling using genetic algorithm. Em *31st International Conference on Information Technology Interfaces, ITI2009*, 2009.

[28] El-Ghazali Talbi e Benjamin Weinberg. Breaking the search space symmetry in partitioning problems. *Theor. Comput. Sci.*, 378(1):78–86, Junho 2007. URL: http://dx.doi.org/10.1016/j.tcs.2007.01.023, doi:10.1016/j.tcs.2007.01.023.

[29] P.C. Chu e J.E. Beasley. Constraint handling in genetic algorithms: The set partitioning problem. *Journal of Heuristics*, 4(4):323–357, 1998. URL: http://dx.doi.org/10.1023/A:1008668508685, doi:10.1023/A:1008668508685.

[30] James C. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6(2):154–160, 1994. URL: http://dx.doi.org/10.1287/ijoc.6.2.154, doi:10.1287/ijoc.6.2.154.

[31] José Fernando Gonçalves e Mauricio G. C. Resende. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, 17(5):487–525, 2011. URL: http://dx.doi.org/10.1007/s10732-010-9143-1, doi:10.1007/s10732-010-9143-1.

[32] José F. Gonçalves, Mauricio GC Resende, e Rodrigo F. Toso. Biased and unbiased random-key genetic algorithms: An experimental analysis. *AT&T Labs Research, Florham Park*, 2012. URL: http://www.optimization-online.org/DB_FILE/2013/01/3741.pdf.

[33] Villiam M. Spears e Kenneth A. De Jong. On the virtues of parameterized uniform crossover. Em *In Proceedings of the Fourth International Conference on Genetic Algorithms*, páginas 230–236, 1991.

[34] Mauricio GC Resende. Introdução aos algoritmos genéticos de chaves aleatórias viciadas. 2011. URL: http://www.din.uem.br/sbpo/sbpo2013/pdf/arq0113.pdf.

[35] Mauricio G. C. Resende, Rodrigo F. Toso, José Fernando Gonçalves, e Ricardo M. A. Silva. A biased random-key genetic algorithm for the steiner triple covering problem. *Optimization Letters*, 6(4):605–619, 2012. URL: http://dx.doi.org/10.1007/s11590-011-0285-3, doi:10.1007/s11590-011-0285-3.

[36] Renata M. Aiex, Mauricio G. C. Resende, e Celso C. Ribeiro. Ttt plots: a perl program to create time-to-target plots. *Optimization Letters*, 1(4):355–366, 2007. URL: http://dx.doi.org/10.1007/s11590-006-0031-4, doi:10.1007/s11590-006-0031-4.

[37] Jose Fernando Goncalves, Jorge Jose de Magalhaes Mendes, e Mauricio GC Resende. A hybrid genetic algorithm for the job shop scheduling problem. *European journal of operational research*, 167(1):77–95, 2005.

[38] José Fernando Gonçalves. A hybrid genetic algorithm-heuristic for a two-dimensional orthogonal packing problem. *European Journal of Operational Research*, 183(3):1212–1229, 2007.

[39] Rodrigo F Toso e Mauricio GC Resende. A c++ application programming interface for biased random-key genetic algorithms. *Optimization Methods and Software*, 30(1):81–93, 2015.